(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 565 314 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
17.01.2001 Bulletin 2001/03

(51) Int Cl.[7]: **G06F 1/00**, G06F 17/60, H04L 9/32

(21) Application number: **93302613.0**

(22) Date of filing: **01.04.1993**

(54) Method for signing travelling programs

Verfahren zur Signierung von wandernden Programmen

Méthode pour signer des programmes itinérants

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU NL PT SE

(30) Priority: **06.04.1992 US 863552**

(43) Date of publication of application:
13.10.1993 Bulletin 1993/41

(60) Divisional application: **00112426.2 / 1 031 908**

(73) Proprietor: **Fischer, Addison M.**
Naples Florida 33942 (US)

(72) Inventor: **Fischer, Addison M.**
Naples Florida 33942 (US)

(74) Representative: **Kuhnen & Wacker**
Patentanwaltsgesellschaft mbH,
Alois-Steinecker-Strasse 22
85354 Freising (DE)

(56) References cited:
US-A- 5 040 142

- **OFFICE AUTOMATION. CONCEPTS AND TOOLS. 1985 , SPRINGER-VERLAG, BERLIN, DE pages 113 - 133 J. HOGG 'Intelligent Message Systems'**
- **THE COMPUTER JOURNAL vol. 33, no. 4 , August 1990 , CAMBRIDGE, GB pages 290 - 295 XP000159499 C. MITCHELL ET AL 'A Secure Messaging Architecture Implementing the X.400-1988 Security Features'**

EP 0 565 314 B1

Printed by Jouve, 75001 PARIS (FR)

Description

FIELD OF THE INVENTION

[0001] The present invention relates to a method and apparatus for creating a "travelling" program which has the capability of moving itself together with necessary associated data from one computer user to another to thereby create a powerful tool for processing, authenticating, and collecting data at various computer nodes.

BACKGROUND AND SUMMARY OF THE INVENTION

[0002] Within an organization, documents are often moved manually. A mail or delivery service is often employed when documents are required to be transmitted between organizations.

[0003] Techniques for electronically transmitting documents within an organization and between organizations are well known. The rapid growth of electronic mail systems, electronic transfer systems and the like have served to automate certain business transactions and eliminate some of the manual document transfers that are in most instances unnecessary.

[0004] One prior art methodology for automatically transferring information between users (for example, within an organization) utilizes a so-called "electronic forms" methodology. This "electronic form" methodology presents data to a user, solicits the user's input via a conventional display, verifies that the input data has been correctly entered, and thereafter transmits such data to another user.

[0005] The electronic form methodology is very limited in many respects. For example, if the data represents any value, then there is always the potential danger that data could be manipulated or altered, or simply, created bogusly. Attempts to address this danger have involved flagging certain critical fields which are to be specially handled. However, it does not permit complex data structures to be assembled and then digitally signed.

[0006] The publication "Office Automation", concept and tools, 1985, Springer Verlag, Berlin, pages 113-133, in an article "Intelligent Message Systems", by J. Hogg, discloses, as an example of an intelligent message as an active object, the printing of queries at a user's terminal, collecting responses and processing such responses in the sense of requesting additional information from recipients in dependancy from a predetermined processing result, so that said processing result determines the way of the message to additional recipients.

[0007] This known system does not provide for any measures of approval, verification of approval or verification of authorisation on the side of the user's terminal and on the side of the recipient, respectively.

[0008] US-patent 5,040,142 discloses a communication system for circulating an electronic document among a plurality of terminals and providing for sequentially adding attest patterns indicative of the approvals of individual reviewing persons to the electronic document. The attest patterns are added to the transmitted document and are kept separate therefrom, so that they can be removed and replaced by another attest pattern of the respective next reviewing person in a sequence of recipients of the transmitted electronic document.

[0009] It is an object of the present invention to provide a method for creating, supporting and processing travelling programs with a remarkable reduction of potential danger of manipulation or alteration or even inadverted falsification on their way from one user to another user or different other users or recipients in the system.

[0010] This object, in accordance with the present invention, is achieved by the features of claim 1.

[0011] Advantageous embodiments and further developments are subject matter of the claims dependent from claim 1.

[0012] The present invention allows for the travelling program to compute, according to any algorithm whatsoever, the digital material which is to be signed, and also as needed, the digital material which is to be verified.

[0013] Thus, for example, the present invention allows the actual data which is signed to be different than any field data itself. In fact, it is possible that the signed material contains none of the actual data as presented by the user.

[0014] An example, of one way this is especially useful is when the travelling program of the present invention creates an EDI (electronic data interchange) transaction based on aspects of the entered data. The program has the ability to sign the EDI transaction. Such EDI transactions may well be composed of complex digital information which was looked-up, based on internal tables within the program, from other tabular files, or from the supervisor or interpreter which drives the travelling program. Thus, input fields which may have been simply entered as "X"s which selected form some table, the actual digital material which is signed is entirely different.

[0015] It is anticipated that the type of digital signature described above may be applied to data construction which will have a long life -- and perhaps be verified by different entities over a period of time. In the case of EDI, for example, the signatures can be bound to the EDI transaction itself, and may be verified by any future recipients of that transaction, even outside the context of the travelling program. This type of digital signature is analogous to a hand-written signature which appears at the bottom of a paper purchase order or contract.

[0016] In addition to being able to sign arbitrary data, the present invention also allows the program to conditionally

decide, based on any known criteria, which users should participate in the signature process.

[0017] For example, with the present invention, the travelling program can make logical determinations, within the program, as to what co-signature requirements may exist for particular data, user, or some combination. This can include information contained in a user's X.500 certificate, or enhanced digital certificate (e.g., as according to the inventor's U.S. Patent No. 4,868,877 or 5,005,200). Because complete programmatic flexibility exists, such extracted information can even be used to regulate the future transmission route for the travelling program.

[0018] In addition to using digital signatures for simple authentication, the present invention also allows authority requirements and uses to be included and verified as well. This draws upon, for example, the teachings of 4,868,877 and 5,005,200 to control authority proof and delegation.

[0019] On the other hand, the present invention also allows uses digital signatures to allow the travelling program to provide other types of valuable authentication. For example, as a security convenience the present invention allows for the digital signature authentication of the entire transmission from one user to another. This includes the travelling program itself, its variables, and any ancillary data or files.

[0020] This second type of digital authentication differs from the data-oriented authentication described above, in part, in that it carries long-term significance -- since the variables and other data which are transmitted will be changed once the receiving user has taken any action at all. This second type of authentication is therefore primarily seen as a protection against tampering, and can also be used forensically as a backward audit to detect unauthorized tampering even by one of the actual users of the form.

[0021] In addition, the present invention also provides a third type of authentication, whereby the travelling program itself may be signed, authenticated and authorized by some trusted issuing authority (e.g., perhaps the author), to insure that no bugs or "viruses" have been introduced. (This even protects against infection by a user which has valid possession of the program along the route).

[0022] The present invention provides a unique mechanism for automating data collection among a group of users. The travelling program may be sent to one user, attach (or detach) relevant data files and move on to the next user. Data or files, collected from one or more users can be deposited with another user, or accumulated for batched processing as desired. This methodology eliminates the need for individual users to be counted on to transmit all the required data in the required format.

[0023] The present invention also efficiently performs electronic document interchange (EDI) in the context of a travelling program which sends itself from user to to the next within an organization, collecting, editing and approving data. At the appropriate point, as determined by the program's logic, it is then able to programmatically generate a standard EDI transaction (e.g., such as the X12 850 Purchase Order transaction set) for transmission to another organization. The travelling program is able to digitally sign the finished transaction set. Accordingly, any receiving organization which can process the standardized EDI, and the standardized signature will be able to authenticate and process the incoming material, even if the receiving organization does not have all the powerful techniques available which are taught by the present invention.

[0024] Conversely, the present invention allows a travelling program to receive ordinary EDI transaction, possibly signed, and allows them to be parsed and incorporated into its variables. The travelling program then has the capability of validating the input and incorporating them into displays, and to move them among various recipients as necessary.

[0025] According to a first aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of program instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; computing a digital value, the content of which is based on logical decisions and manipulations performed by said program; and performing a digital signature on said digital value at at least one destination.

[0026] According to a second aspect of the invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; acquiring data from users of at least one of said computers via execution of said instructions; translating said data via the executing of said instructions into a specialized data structure conforming at least in part to a recognized standard whereby said data structure is useful independently of said instructions; and digitally signing said data structure via the execution of said instructions.

[0027] According to a third aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a computer with a first travelling program comprising

a sequence of instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; providing at least one of said computers with a second travelling program; executing the second travelling program under direction of the first travelling program.

[0028] According to a fourth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a computer with a first travelling program instance comprising a sequence of instructions which are executed by the computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; providing at least one of said computers with a second travelling program instance; processing the second travelling program under direction of instructions in the first travelling program instance.

[0029] According to a fifth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; and selecting a file in response to execution of said sequence of instructions; transmitting at least part of the content of said selected data file to said next destination in response to execution of said sequence of instructions.

[0030] According to sixth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for forwarding information in said communications system comprising the steps of: providing a first computer with a set of instructions which are executed by the first computer including instructions which generate a plurality of instances of said set of instructions and which initiate transmission to at least a first and a second destination which respectively receive one of said instances together with accompanying data; and including within said instances transmitted to said first and second destinations the capability of subsequently merging data that has been accumulated during their distinct transmission paths.

[0031] According to a seventh aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of program instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; and qualifying the set of operations which said sequence of instructions is allowed to perform.

[0032] According to an eighth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of program instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; and performing a digital signature by using a private key stored in a user token device.

[0033] According to a ninth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method for processing information among said computers comprising the steps of: providing a first computer with a sequence of program instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination; and performing a date/time notarization.

[0034] According to a tenth aspect of this invention, there is provided in a communications system having a plurality of computers coupled to a channel over which computers may exchange messages, a method of processing information among said computers comprising the steps of: providing a first computer with a sequence of program instructions which are executed by the first computer, including instructions which determine at least one next destination that should receive the set of instructions, said set of instructions including instructions for transmitting said instructions together with accompanying data to said next destination. and performing a time delay function.

BRIEF DESCRIPTION OF THE DRAWINGS

[0035] These as well as other features of this invention will be better appreciated by reading the following description

of the preferred exemplary embodiment of the present invention taken in conjunction with the accompanying drawings of which:

FIGURE 1 is a block diagram of a communication system in accordance with an exemplary embodiment of the present invention;

FIGURE 2 shows an exemplary structure of a travelling program together with its associated components;

FIGURE 3 shows an exemplary execution control area data structure;

FIGURE 4 shows the data structure of a file control block (FCB) which is used when a travelling program attaches files to, or detaches files from itself;

FIGURE 5 shows a process control block that is utilized in the execution of a travelling program;

FIGURE 6 illustrates a variable control block data structure (VCB) which is used for controlling variables;

FIGURE 7 shows an exemplary travelling program loader;

FIGURE 8 shows how the header is loaded;

FIGURE 9 shows how the "program" segment of the travelling program is loaded;

FIGURE 10 shows how the "variables" segment of the travelling program is loaded;

FIGURE 11 shows how the "certificate" segment of the travelling program is loaded;

FIGURE 12 shows how the "file" segment of the travelling program is loaded;

FIGURE 13 delineates how the "closure" segment of the travelling program is loaded;

FIGURE 14 represents the operations performed in processing P-code instructions;

FIGURE 15 shows processing which takes place after the P-code operation is performed;

FIGURES 16A and 16B show processing for handling program defined functions or calls;

FIGURE 17 shows the sequence of operations for handling built-in functions;

FIGURES 18 and 19 delineate the sequence of operations performed for executing external functions or calls;

FIGURES 20 and 21 delineate the operations which are performed when a travelling program mails itself to a predetermined recipient;

FIGURE 22 delineates the sequence of operations for attaching a file to the travelling program;

FIGURE 23 shows how a file may be erased from a user's system;

FIGURE 24 shows the sequence of operations performed in detaching a file from a travelling program;

FIGURE 25 delineates the sequence of operations performed when a file has been transformed into a user file;

FIGURE 26 delineates the sequence of operation performed when material is to be digitally signed;

FIGURE 27 delineate the sequence of operation performed by a "INTER-ROLLOUT" function;

FIGURE 28 shows the sequence of operations performed when displaying information to the user;

FIGURE 29 delineates the sequence of operation performed by the "time delay" routine;

FIGURE 30 shows the sequence of operations for a "select from directory" function;

FIGURE 31 is a routine which demonstrates how the the interpreter program permits a user to perform digital signatures;

FIGURE 32 exemplifies how a user verifies received information;

FIGURE 33 illustrates how a travelling program collects a file to be transferred;

FIGURE 34 illustrates the travelling program operations performed in reading data from a specified file;

FIGURE 35 illustrates how the travelling program may update or create a file from program variables;

FIGURE 36 illustrates how a travelling program may be designed to be split and send programs to a number of different recipients;

FIGURE 37 demonstrates how previously split programs may be merged;

FIGURE 38 shows an alternative approach to merging previously split travelling program information;

FIGURE 39 is a flowchart indicating how the travelling program has been designed to accommodate electronic document interchange generation functions; and

FIGURE 40 relates to the use of travelling program in receiving an electronic data interchange transaction.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0036] Figure 1 shows a block diagram showing an exemplary communication system which may be used in conjunction with the present invention. This system includes a communication channel 12 over which communication between terminals A, B,..N, may take place. Communication channel 12 may, for example, be an unsecured communications channel such as a telephone line.

[0037] Terminals, A, B, ...N may, by way of example only, be IBM PC compatible computers, having a processor (with main memory) which is coupled to a conventional keyboard/CRT display 4. The processor with main memory 2 is also coupled to a non-volatile storage which may be a disk memory. Each terminal, A, B...N also includes a conventional IBM PC communications board (not shown) which, when coupled to a conventional modem (6, 8, 10, respectively), permits a terminal to transmit and receive messages including travelling programs.

[0038] As used herein, a "travelling program" is a digital data structure which includes a sequence of instructions and associated data and which has the capability of determining at least one next destination or recipient for receiving the travelling program and for transmitting itself together with all relevant data determined by the program to the next recipient or destination.

[0039] Each terminal is capable of generating a message and performing whatever digital signature operations may be required to load and execute the logic, data, and functions inherent within the travelling program (as described more fully herein), and transmitting the message to other terminals connected to communication channel 12 (or to a communications network (not shown) which may be connected to a communication channel 12).

[0040] The digital signature and certification methodology described in the inventor's U.S. Patent Nos. 4,868,877 and 5,005,200, as well as 5,001,752 may be used herein, which patents are hereby expressly incorporated herein by reference. Alternatively, more conventional digital signature methodology may be utilized.

[0041] Before describing the details of the "travelling program" structure and methodology in accordance with an illustrative embodiment of the present invention, an example of the general operation in an actual business transaction context will be briefly described. Initially, presume that the user of the Figure 1 terminal A is a relatively low level engineer who is a part of a design team in a corporation seeking to obtain component parts to complete a circuit design project.

[0042] The engineer using keyboard 4 would access a parts requisition "travelling program" of the type to be described in detail below. The requisition "travelling program" will prompt the engineer to describe the component parts needed. The travelling program includes an instruction sequence which will automatically transmit itself to a next destination, e.g., to a supervisor who has access to terminal B and who is higher up in the organizational structure and

possesses the authority to approve the requisition request and digitally sign it. The travelling program may also transmit ancillary information, such as files which may be necessary or useful at future destinations. The supervisor will be prompted to properly digitally sign the request. It is possible that the digital signature reflects not only specific variables values, but also the variable names. Alternatively, the signature may also reflect some aggregate structure which is derived from variables computed within the program, wherein the values may be based on any of many sources, including data read from file, user input, data built into the program, various signer's certificates, or data which is extracted from the user environment (such as the user's ID), etc.

[0043] If the request is approved, the requisition form will take a different path in the organization then if it is not approved. The travelling program can have the intelligence to determine, based upon the input from the supervisor at the operating terminal B, where to transmit itself within the organization. The travelling program will also, if desired, load the memory associated with terminal B with the appropriate data relating to the requisition and to attach if desired any files from terminal B that needs to be forwarded elsewhere in the organization.

[0044] Once a signature has been done, the travelling program has the ability at any later time, for any later user, for any reason to recompute any material to be verified, and to perform a digital signature verification.

[0045] The results of such verification can be announced to any recipient, or more likely, the travelling program can simply perform the verification and announce a problem should there be a failure (which suggests attempted data tampering).

[0046] Because the travelling program monitor may embody the teachings of 4,868,877 and 5,005,200, it is possible for authorization to also be checked so that any recipient can be assured that the necessary authorizations were performed.

[0047] After a particular data structure has been constructed and signed under control of the travelling program, it is possible to subsequently reconstruct that data structure and to provide its signature to any other entity. Such data can not be subsequently tampered by any entity.

[0048] However, the present invention also embodies capability whereby all the transmitted data is digitally signed as it is sent from one user to the next. The travelling program processor in the recipient's computer can automatically verify this signature as the travelling program is loaded. This assures that no component whatsoever is altered or tampered along the way. While this overall signature only reflects the state of the data during this particular transmission, and has no significance for later users, it does insure a perfect transmission untampered by third parties, and it does provide a forensic audit mechanism if it is necessary to trace covert tampering by participating users, while those users had possession of the form. This overall signature differs from current capabilities whereby electronic mail is signed, in that the signature can be conditionally induced by the travelling program itself, as part of the transmission process.

[0049] Ultimately, after all the approvals have been obtained within the organizational structure, the travelling program will create an actual Purchase Order.

[0050] This could be done in many ways. It may well be possible for a travelling program to support several methods, choosing the one most appropriate for a given circumstance. We describe four possibilities here:

1. The travelling program could simply print out the final purchase order on paper -- possibly even printing the company logo, letterhead, etc. -- which would be physically mailed.

2. The travelling program, if coupled with an outgoing computer-to-fax capability, could automatically generate a purchase order image, that would appear on the vendor's fax machine. The buyer would not have to produce paper.

3. If it is known that the vendor also supports the travelling program methodology of the present invention, then it is possible that the travelling program will simply designate the vendor as a next destination.

4. It is also very possible that the vendor does not use the present invention, or that the purchaser's travelling program cannot determine with certainty that the vendor is able to handle the travelling program methodology.

[0051] Therefore, the travelling program manipulates its internal data to construct a standardized EDI (Electronic Data Interchange) transaction, which can be widely recognized and processed. The travelling program may also cause a digital signature to be performed on the computer EDI transaction, and the signature and the transaction can both be transmitted. The travelling program would then transmit the EDI transaction, as well as any possible signature, to the recipient. (Such transmission is independent of, and should not be confused with, the transmission of the travelling program and its ensemble from user to user as part of its directed travels.)

[0052] Any recipient that can handle standardized EDI transactions is then able to handle the received EDI input. Any recipient that can handle digital signatures, is further able to authenticate the transaction. Furthermore, provided the recipient has sufficient software capability to recognize them, the recipient can also automatically validate any authorization that may be embodied as part of the signature. It is up to the logic of the travelling program the extent to

which certificates should be transmitted along with a signed transaction.

[0053] In any of the above cases, the travelling program can spin off the purchase order (P.O.) information to the vendor, using any of several possible levels of automation. Following this, the travelling program might transmit one version of itself, or possibly just a letter, back to the originator, to inform him that the P.O. has been sent. Other information can be sent to an archive, or to a queue to await further processing. This information could be a simple message, a record added to a file, or perhaps the travelling program schedules a full traversal (automatic "mailing" or transmission).

[0054] Figure 2 illustrates the structure of a travelling program together with its associated components in accordance with an exemplary embodiment of the present invention. The Figure 2 travelling program includes at least the following multi-field segments. A first header segment 20 preferably identifies the size of each of the component segments, the name of the associated program (and possibly other segments described below), the date, the type of each component (e.g., the program is the source language program, or the program is P-code that has already been compiled), the identity of the form, version of the interpreter needed to execute it, data necessary to resume execution at the appropriate point of program resumption (such as execution stacks, PCBs, etc.), dates associated with the latest traversal, and program authorization information (PAI). Each segment in the travelling program structure may include its own description so that the "type of each component and size" field "S" would not be included in the header segment 20. For the purposes of the present application, program authorization information (PAI) may be regarded as security information which defines the range of operations that the associated program is permitted to perform (e.g., defining access to files, the ability to call programs, ability to generate electronic mail, ability to transmit data to other users, ability to release documents, ability to execute machine language programs, ability to access special areas of memory, ability to display information to users, ability to solicit digital signatures, ability to access a digital notary public device, etc.). Further details regarding the nature and use of the program authorization information may be found in US 5,412,717 entitled, "Computer System Method and Apparatus Using Program Authorization Information (Atty Dkt. 264-29). The header segment 20 may also include a version number of the associated travelling program.

[0055] The travelling program code 22 segment follows the header in the exemplary embodiment and preferably is written in the restructured external execution programming language (e.g., the REXX language) or something akin to PASCAL or COBOL. The program itself may, for example, relate to a purchase order related application.

[0056] The travelling program will possess the characteristics described above including the ability to transmit itself to further recipients. Thus, program 22 will include instructions for forwarding itself via whatever medium is available to one or more recipients this is known herein as a "traversal". One source code instruction or several P-code instructions may be required to result in the "traversal" of the travelling program to one or more identified recipient(s). The travelling program structure set forth in Figure 2 is designed to be independent of any particular computer architecture and is structured in accordance with international standards (e.g., X.209 format).

[0057] The travelling program also includes a "variables segment" 24. Prior to being executed by a first user, the variable segment 24 may be virtually empty. Once the program is sent to a recipient, further variables will become defined as they are required by the program to thereby result in an increasing number of variables as the program is further executed. By way of example only, the variable section 24 may identify a variable, such as "total.dollars.received" together with an actual data value for this variable.

[0058] Each variable may have associated therewith the information set forth in each of the fields 32-42 shown in Figure 2. Field 32 identifies the size of the variable name. The variable name itself is stored in field 34. The size of the value of the variable is set forth in field 36. The value of the variable is in field 38. Field 40 identifies the execution stack level to which the variable belongs. The execution stack level is identified since the same variable name can exist at different levels within a program (e.g., one variable name may exist in a first subroutine while the same variable name may exist in a separate or nested subroutine and yet have a different definition). The execution stack level is helpful in reconstructing the travelling program in a recipient's computer to take on the same logical structure it had in the sender's computer. Field 42 is an optional field which may identify a type of variable, e.g., strings, octets, integers, etc.

[0059] The "variables" section 24 may also include a digital signature of the respective variables and related information. Thus, it is also possible for one or more variables to reflect digital signatures which have been taken at various times during the travelling program's execution path. One of the significant aspects of the current invention is that the travelling program can create a digital signature on any type of information. This signature is itself carried as a variable. To verify the signature it is necessary for the program io indicate (or possibly re-compute) the exact value which was signed, and then pass that, together with the signature value (also indicated by a variable) to the VERIFYSIGNATURE function of the travelling program. By including a digital signature of variables, a recipient will be enabled to verify that the data 1) has not been tampered with, 2) has been validly signed, and 3) the signer was properly authorized. See above identified U.S. Patent No. 5,005,200, which describes a preferred mechanism for associating authority with a digital signature.

[0060] A segment 26 is shown in Figure 2 for optionally including with the travelling program, certificates associated with any digital signatures so that any signatures may be verified by a recipient as described, for example, in the above-

identified U.S. Patent No. 5,005,200. Alternatively, the certificates could be included in the "variables" section together with the digital signatures.

[0061] Segments 28A-28N contain file images that are recorded and tagged by name to enable the travelling program to attach and store a file belonging to a travelling program user. Thereafter, the user's file may be transmitted along with other prior user's files with the travelling program. The name of the file facilitates later accessing of the file by a user and permits the travelling program user to identify any file which is, for example, to be further transmitted, or which is to be deposited with a particular user under particular circumstances.

[0062] The travelling program also includes a "closure segment" 30 which includes, for example, the digital signature of the entire travelling structure so that the recipient can verify that the transmission of the entire travelling structure has not been tampered with since it was last sent.

[0063] Having described the travelling program data structure, we now describe the data structures utilized during the execution of a travelling program and the associated software for executing the travelling program. An execution control area (XCA) data structure is shown in Figure 3. The XCA specifies information required by the program which executes the travelling program, once the travelling program has been received by a recipient, and compiled into P-code (unless it was originally transmitted in P-code).

[0064] As shown in Figure 3, XCA segment 82 identifies the address and size of the program as it appeared in the incoming file. It should be recognized that, throughout this description, whenever a segment is stated as storing an "address" or "location", that the data may be a physical or logical address and need not necessarily directly specify an actual physical memory location. The program may be received in source or P-code and an indication is maintained as to which is the case. The execution control area includes a segment 84 which is indicative of the address of the p-code version of the program and its size. The address (or pointer to the address) of the current program control block is identified in segment 86. The location of the list of file control blocks (FCB) which is used, for example, to attach and detach files associated with the travelling program is set forth in segment 88. The address of the certificate control area (CCA) which is used for controlling certificates which are attached to the travelling program is set forth in segment 90. The location of the "variable" information table (VIT) is set forth in segment 92 which controls and maintains variables in the form of a "B-tree", which is a hierarchical binary tree structure which identifies the location of each program "variable".

[0065] The execution control area also includes a security information segment 94 which may be used for verifying the authenticity and the authority implicit in the travelling program. Segment 96 defines the name of the file that contains the incoming travelling program which may need to be accessed. Segment 98 keeps track of the number of times the program has mailed itself along the incoming path. The execution control area also includes an input parameter section 100, whereby parameters relating to the execution of the program may be identified. Execution control area segment 102 identifies the input header information received from the travelling program file so that the header information will be available.

[0066] Figure 4 shows the data structure of a file control block (FCB) which is used when a travelling program attaches files to, or detaches files from itself. The file control block includes a tag field 116 which identifies a tag for referencing a particular file to be attached or detached in a particular user's system. The file control block also includes a segment 110 which is a pointer to the next file control block. The file control block also includes a status segment 112 which defines various status conditions such as whether the associated file has just been attached by the received travelling program; whether the file can be detached on the next traversal (i.e., next mailing); whether the file has been exported (i.e., the associated file image has already been loaded into a separate user file); and an indicator as to the "type of file" such as whether the file is stream oriented or record oriented. Other attributes of the file may be defined in this field.

[0067] Segment 114 stores an indication as to the file's position within the main incoming travelling program file so that the particular file in question may be quickly accessed. Segment 118 identifies whether the local name of the file (i.e., the file name identified by the most recent recipient of the travelling program). The local name of the file is typically provided if the file has been attached and is being forwarded to a further recipient or if an already attached file is being "exported", i.e., stored locally by a particular user. Additionally, as shown in Figure 4, the FCB may contain a hash of the associated file. As will be appreciated by those skilled in art, a hash is a "one way" function which should be computationally easy to compute given the underlying data. The hash function should be computationally impossible given a hash value, to either determine the underlying data, or to create any data which has the specified value as its hash. For all practical purposes, the value obtained from applying a hashing function to the original aggregation of data is an unforgeable unique fingerprint of the original data If the original data is changed in any manner, the hash of such modified data will be different.

[0068] Figure 5 shows an exemplary program control block that may be used during the execution of the travelling program. A program control block keeps track of control information regarding the program being executed in a structured programming context where one routine calls another routine, each routine having an associated program control block.

[0069] The program control block segment 50 points to the prior program control block in the program execution

control stack. The program control block includes a segment 52 which defines the next P-code position to be executed in the current executing program and segment 54 defines the type of last P-code operation performed. Segment 56 includes a pointer to an expression evaluation stack which is used during expression evaluation. The execution stack is typically distinct from the program stack, in that the execution stack is used for evaluating expressions and keeping track of internal state. Segment 58 defines the level of this stacking program and segment 60 defines a pointer to a list of shared variables. In the REXX language an "exposed" statement may be used for accessing shared variables.

[0070] Figure 6 illustrates a variable control block data structure (VCB) which is used for controlling variables. Segment 62 identifies where in the B-tree a variable is located and may contain several pointers. Segment 64 identifies the size of the variable value and segment 66 identifies a pointer to where the value is located in memory. Segment 68 may be optionally used to identify the type of variable. Segment 70 identifies which level of the travelling program the variable is associated with, so that after the program is executed, any local variable which was associated with the program may be readily deleted. Segments 76 and 80 identify the size of the variable name and the name, respectively.

[0071] We now turn to illustrating the execution of the travelling program. The sequence of operations performed by a "loader" portion of an interpreter execution-driving program is set forth in Figures 7-12. These operations relate to preparing to execute a travelling program.

[0072] A travelling program may execute in one of a plurality of different modes such as an interactive user mode, a mode in which it is called by another program, or a batch operation mode in which it is sent from node to node collecting information. Initialization information is input during the start-up operation (120) to identify the particular operating mode as well as associated run-time parameters.

[0073] The flowcharts set forth in Figures 7-12 illustrate how a travelling program structure shown in Figure 2 is loaded. In loading the travelling program, the interpreter creates the execution control area XCA and initial program control block PCB. It saves access to input parameters, saves the names of the input files that it has been given to load and initializes the variable information table (VIT) (122). In flowchart block 122, the execution control area and program control block associated with the travelling program are established. The various XCA and PCB fields are filled in during subsequent processing.

[0074] Thereafter, the loader begins loading travelling program segments, i.e., header, program, variables, certificates, file and closure segments as shown in Figure 2. Loading each of the travelling program segments described above, e.g., header program, etc., causes appropriate data to be filled in as described below.

[0075] In block 124, a decision is made as to whether more segments need to be processed. If so, then the initial input is read for that segment and the type of segment is determined after which segment processing is initiated depending upon the type of segment (126).

[0076] Turning to the header processing of Figure 8, initially, a check is made to determine whether the segment being processed is the first segment (150). If not, then an error condition exists (152) since the header must be the first segment. If the first segment is being processed, then the header is read and hashed. The header data is stored into the XCA (154).

[0077] The routine then branches back to Figure 7 at entry point L. The loader determines whether there are any more segments to be processed (124). If so, block 126 is executed to result in the processing of the "program" segment as shown in Figure 9. Initially, a check is made to determine whether there is a header, and no program has yet been loaded (160). If the answer is no, then an error condition exists (162). If the answer is yes, then the program is read and a hash is taken (164).

[0078] Thereafter, the program hash and/or digital signatures associated with the program (and/or the header) are verified 166. If the digital signatures were not properly authorized or could not be verified, then an error condition results 166. If verification occurs, then any security and authorization information associated with the travelling program is saved (170). Such authorization information could alternatively be kept in the header or in the program segment.

[0079] In block 172, a check is made to determine whether the program has been sent as P-code. If source code rather than P-code has been sent, then the source code is compiled into P-code using conventional compiler techniques known to those skilled in the art and the source code image is deleted from storage 174. Regardless of the check at block 172, the position in the incoming file of the program -- whether it is in source or P-code format -- is saved in the XCA. Knowing the location and extent of the incoming image simplifies the copying of the program into eventual outbound traversal(s). Finally, regardless of whether the P-code was just compiled, or whether it was read form the incoming file, the main storage address and size of the P-code is set into the execution control area (XCA) in 178, after which the routine shown in Figure 7 is reentered at block 124 to thereby result in loading remaining segments such as the "variable" segment processing shown in Figure 10.

[0080] In processing the "variable" segment as indicated in block 190, a check is made to determine whether the header and program have been loaded but no prior variables. If this is not the case, then an error condition results 192. If a header and program have been loaded, but no prior variables, then we begin an iterate process to read all the variables, if any. A check is made at 194 to determine whether there are (more) variables to read. If there are more variables to read, then for each variable, a variable control block (VCB) is created as shown in Figure 6 and is completed

by the insertion of a variable identifier and value into the variable control block (VCB) and the setting of certain status conditions in the VCB. Additionally, the variable control block is added to the proper spot in the variable information table (VIT), the table which contains all program variables (196).

[0081] Additionally, other variable information, for example, related to previous executions of the travelling program are loaded into memory stacks or program control blocks as appropriate 198. Alternatively, it may be desirable to keep such "control" information in the header segment rather than here. Thereafter, the routine branches back to block 194, where checks are made to determine whether more variables are required to be read. The processing continues until no more variables need to be read, at which point the routine branches back to block 124 of Figure 7 to thereby result in loading the next segment.

[0082] As indicated in Figure 11, each certificate is read (200) and a certificate element is created which is then added to a certificate control area (CCA) in storage (202). As schematically indicated in Figure 11, the process is repeated until all certificates are received at which point the routine branches back to block 124 to check for any more segments.

[0083] Alternatively, it may be desirable to transmit the certificate segment ahead of the program segment, so that certificates used as part of program authentication/authorization can be maintained together with any certificates used by program variables and user-to-user authentication.

[0084] This branching operation results in the "file" segment processing shown in Figure 12. Since the file segments typically follow the "variable" segments, a check is made to determine whether the variable segment (even if null) has already been loaded. If not, then an error has been detected and an appropriate error message is generated 212. If the "variable" segment has already been loaded, then as indicated in block 214, a check is made to determine whether the file tag associated with the file has already been loaded. If so, then an error is detected indicating that the file has been duplicated 216.

[0085] If the file tag has not already been loaded, then as indicated in block 218, a file control block is built for the file, the tag name is set, other status indicators are set that may have already been associated with the travelling program, and the file position is set relative to the incoming file.

[0086] Thereafter, the file is read and its hash is computed and saved in segment 115 of the FCB. The size of the file is saved in segment 114 of the FCB. The file need not be loaded into memory at this time (220). Thereafter, the file control block which has been created is added to the file control block list collected in the XCA and the routine branches back to block 124 to process the next segment (probably "closure").

[0087] In the "closure" processing in Figure 13, the hash is computed of all previous hashes for each previous segment (230). It should be recognized that all the "segment" material is read subject to hashing. A check is then made in block 232 to determine whether the hash taken and calculated in 230 matches the hash added when the travelling program was sent (which is stored in the closure segment). If there is no match, then an error condition results 234.

[0088] If there is a match, a check is made as to whether the travelling program is signed (236). If not, then as suggested at block 238, an action is taken to incorporate whatever level of security is desired, such as possibly presenting a notification that the transmission data is not entirely signed (238).

[0089] If the transmission was signed, then the signature is verified and a message is presented to the user to accurately identify the party who actually sent the travelling program (and the associated purchase order or other form) 240. The routine then branches back to block 124 of Figure 7.

[0090] The completion of the "closure" processing in Figure 13 results in block 124 determining that there are not more segments to be processed. Thereafter, a check is made to determine whether closure was successfully received and processed (128). If it was not, then the routine stops execution after performing an unsuccessful validity check (130) and processing halts 132.

[0091] If the check at block 128 reveals that closure was successfully completed, then various steps are taken to prepare for program execution (134). In this regard, stacks are restored, the variable information table and variable control blocks are restored. The program control blocks are restored such that they contain the execution resumption point.

[0092] Thereafter, the routine shown in Figure 14 is initiated to actually process the P-code instructions. The following problem must be considered here. Because the program execution is effectively restored identically to the state it was at the time it was transmitted (as part of the traversal) from the sender's machine, there is an issue of how the travelling program can distinguish whether it is in the sending machine, and just returned from the sending itself; or whether it has just been restored in the recipient machine.

[0093] The present invention allows multiple ways to address this problem. If the traversal function is implemented as a built-in function, then the interpreter will return a special value (say "0") to the program after it has successfully sent itself, and another value (say "1") to the program when its execution is restored on the recipient's machine. The travelling program can then test this value to distinguish the situation. Another way this distinction could be made is by providing the travelling program a function to extract the "number of prior traversals" (segment 98 in the XCA). Before invoking the traversal, the program could use this function to save the prior-traversal-count function. If it matches

the value of the variable, then the program knows the execution is resuming in the sender's computer; if it differs (and it should only be one greater), then the program knows the execution is resuming at the recipient's computer.

[0094] When the first user generates the travelling program, the loader routine shown in Figure 7-13 is executed with very few, perhaps no, variables, files, or certificates. Accordingly, certain of the above-described steps will be omitted during the initial processing. The loader routine is executed whether the travelling program is executed for the first time or executed by further recipients.

[0095] Figure 14 illustrates the operations performed in processing P-code instructions; it is repeated for every P-code instruction executed. As indicated in block 250, the location of the next P-code instruction is derived from the current PCB (52), and this becomes the "current" P-code operation. In block 252, the length of this P-code operation is determined, and the "next P-code" position (52) is updated to reflect the subsequent P-code instruction. The type of the current P-code operation is saved in (54) (It is useful for the interpreter to share common routines which have slight variations based on the precise operation. For example, the "call" operation and the "function invocation" operation are similar except that the function invocation expects a parameter to be returned).

[0096] Thereafter, as illustrated in block 254, the indicated P-code operation is performed. Most P-code functions involve data manipulation, logical tests and program flow control. By way of example only, such P-code operations may include locating a variable and pushing the variable in a stack, resetting the next P-code operation to thereby change the flow of control such as would occur in a branching operation, performing an arithmetic or string operation, performing IF/THEN/ELSE operation based on the popped stack value, perform DO/ITERATE/UNTIL/WHILE, or other operations based on stack values, performing SELECT/WHEN/OTHERWISE operations based on the stack values, performing an "END" operation to close a DO/WHEN/SELECT operation.

[0097] We will soon discuss in some detail various P-code operations pertinent to the present invention's unique operation. With the guidance given herein, the P-code functions can be implemented in a straight-forward manner by anyone familiar with writing interpreters.

[0098] However, ignoring for the moment the details of the particular P-code function, the preferred design allows for P-code operations to generate logical "interrupts" at their completion.

[0099] These allow processing P-code processing to be suspended while some other, external operation must be performed. This interrupt concept is used in the preferred design to facilitate the rollout of working storage whenever lengthy waits or external activity is invoked.

[0100] In Figure 15, on return from the P-code routine in block 256, the interpreter determines whether the routine has signaled a logical interrupt. If not, then return is made to 250 to handle the next P-code operation.

[0101] If an interrupt was indicated, a special check in block 258 is made to determine whether this is the special "EXIT" request. If so, then all resources which should be released at the end of this program, such as storage, files, variables, load subroutines, etc., are discarded in block 260. A possible return value from the P-code operation, which may have been saved by 260, is returned in block 259 to the invoker of this travelling program.

[0102] Assuming, this is not EXIT, then block 261 determines whether ROLLOUT should be performed. For example, in certain environments, it is useful for working storage to be rolled out while a user completes entering input, or while the travelling program is waiting for a time interval to expire, or while a lengthy (or large) external program has been invoked from the travelling program logic, or while the digital signature routine is being executed (since that often involves user input).

[0103] Routines which cause a P-code interrupt and a possible ROLLOUT, regardless of whether they are implemented as built-in functions or as language statements (with their own P-code), include:

SIGN	which applies a digital signature to any computer data, and in doing so may solicit the user to select from multiple certificates, and solicit the user to provide his secret password key which allows the private signature key to be decrypted and used;
DISPLAY	compose and output a screen and wait for the user to supply input;
TIMEWAIT	suspend execution until a future time is reached;
SELECT.FROM.DIRECTORY	which allows selection from , e.g., a directory of users, or a directory of files, etc.
NOTARIZE	wait for a time notary device to apply its own digital signature.

[0104] In some environments, ROLLOUT is pointless, and in these cases the rollout and rollin processes in block 262, 264, 268 will be absent or inhibited.

[0105] In any case, a P-code operation which signals an "interrupt" also supplies the address of at least 3 associated ("call-back") functions --

-- the pre-rollout routine, which performs any required functions in preparation for rollout. This might include preparing a parm field in temporary storage to pass to ...

-- the inter-rollout routine which executes after as much working storage as possible has been rolled out to auxiliary storage.

-- the post-wait routine which handles details following the rollback after the inter-rollout routine is finished, and after working storage has been restored from auxiliary. Typically, this involves copying a result value computed by the inter-rollout routine which is left in temporary storage, and which must be loaded onto the execution or copied into a program variable.

[0106] In block 261, the pre-rollout routine is invoked. This may be a null routine, or it may setup, e.g., parameters for the inter-rollout routine.

[0107] In block 262, the rollout function is performed, if appropriate given the environment and circumstance.s If done, then ROLLOUT consists of writing all working storage, including the VCBs and their values, the FCBs, the certificates and the CCA, the execution stack, the VIT, the XCA, the P-code itself, and any other blocks, to some auxiliary storage (such as a file). The interpreter itself may be released from storage, and this may be done in a special block (264), provided that sufficient residual program and data remains to later reload the interpreter and the working storage.

[0108] In step 266, the inter-rollout routine is invoked. Typically, this routine waits for the user to enter input, or to wait until a future time or other event, or to invoke another program which might wait for input, or cause other delays, or require a large of storage which is vacated by the ROLLOUT.

[0109] In block 268, after the inter-rollout is finished, the interpreter is reloaded, then the working storage, including the P-code, the execution stack, all control blocks are restored from auxiliary storage.

[0110] Then in block 270, any final processing is done to tidy up the operation. For example, this typically includes copying a result returned by the inter-rollout routine to the execution stack, or to a program "variable".

[0111] This completes the interrupt, and control is then returned to the top of the P-code handler (250), where the next P-code instruction is processed.

[0112] We now examine some P-code operations of interest.

[0113] The interpreter in the preferred embodiment handles three of CALLs and function: to routines which are "built-in" to the interpreter, to routines which are written as part of the travelling program, and to routines which are external to the interpreter or program, and which are dynamically located and invoked when the program is executed.

[0114] In Figure 17 we see that the built-in function appears rather simple, and the interpreter simply locates the specified function based on an index in the P-code, and lookups the routine's address (within the interpreter), and calls it. However, it is important to realize that, while most do not, some built-in functions might signal a P-code interrupt. In this case, the built-in function must provide the necessary pre-rollout, inter-rollout and post-wait routines.

[0115] The P-code interpreter always distinguishes CALL and functions, and provides for the return of a result to the execution stack in and only in the case of a function. For example, the SIGN function returns a value which represents the digital signature computed on the supplied data.

[0116] In Figure 16A we see that a call/function to a program routine causes the creation of a new PCB execution level 300. The new PCB is set to start executing at the start of the subroutine, by setting the next-P-code instruction (52) to the P-code entry point of the routine. The first instruction of the routine will be accessed when block 250 is reentered. Parameters are prepared for the program routine, appropriate status condition are set, the program level 58 in the PCB is set to one higher than the calling program and the PCB is placed at the top of the execution stack as the now current PCB (82). The result of a program routine is passed to the caller through the P-code RETURN operation.

[0117] In Figure 16B, we see how the corresponding program RETURN P-code operation operates. Block 1200 determines if a RETURN is made from the highest (only) level PCB, in which case this operates as an EXIT, and block 1204 signals that a P-code "EXIT" interrupt is required and passes the return RESULT (if any) as the value to eventually be returned by block 261 (Fig. 15) as the RESULT for the entire program.

[0118] Otherwise, in block 1204, determination is made as to whether the invoker used a CALL or function (e.g., by checking field 54 in the caller's PCB), and in the latter case block 1206 puts the return VALUE on the stack (or creates a default value if the RETURN had no operand).

[0119] In block 1208, the current level is cleaned-up, and all resources, including storage, files, variables, etc private to this subroutine (aka "program level") are released. Resources, such as variables which are shared with the caller are NOT released arid are available.

[0120] In block 1210, the current PCB is then released so that the caller's PCB now becomes the current one, and return is made to block 256 where execution resumes.

[0121] The interpreter includes built in routines which are designed to accomplish specialized travelling program

related functions relating to providing digital signatures, user files to the travelling program and other functions to eliminate the need for a travelling program designer to be concerned with programming such functions.

[0122] P-code operations may also involve the performance of a RETURN function which will affect program control, a PROC operation which relates to a program control block. The interpreter also performs a DISPLAY operation which utilizes the interactive display methodology/language described herein. The interpreter also performs a TRAVERSE operation which results in the "mailing" of the travelling program to another recipient as well as all associated data.

[0123] Figure 18 illustrates an exemplary the sequence of operations performed for executing external functions or calls. Such external functions or calls are not built in to the interpreter or part of the travelling program but rather are part of the user's program library. The named function or call is located from any of several possible libraries 354.

[0124] A check is then made to determine if the program is found 356. If the program is not found, then a check may, if desired, be made to determine whether the program should be terminated or some default action be performed 358. If a decision is made to terminate, then an error message is generated, and after various housekeeping/cleanup operations are performed as described above the program is exited (360, 362).

[0125] If the check at block 358 indicates that a default action should be taken, then the default action is taken, e. g., by returning a special default function value (368) and the routine branches back to node 0 in Figure 14 to begin executing further P-code instructions.

[0126] If the program is found as a result of the check made in block 356, then parameters are constructed by the program (364). Invoking external routines involves a P-code interrupt, with a possible rollout. This allows us to conserve storage in multi-user swapping environments if the external program is lengthy, or in any environment if the external routine is huge and therefore the storage used by the travelling program should be vacated in order to satisfactorily perform the external program. In this case, the P-code interrupt is signaled in block 366. The indicated PRE-ROLLOUT routine copies the parameters to the external form the stack (or variables) to temporary storage. The INTER-ROLLOUT routine invokes the EXTERNAL routine and receives any returned result; and the POST-WAIT routine copies the returned result to the stack (if the external routine was invoked as a function).

[0127] It is possible that the external routine is actually another travelling program. If so, then special optimization may be performed by using the existing already-loaded image of the P-code interpreter, and simply passing a new set of parameters to block 120 (Fig. 7). In this, special logic would need to be inserted in blocks 262 and 264 to conditionally avoid releasing the interpreter code itself.

[0128] Now let us turn out attention to various special built-in functions which are used by the present embodiment. Many of these could be executed either as built-in functions, or as language statements with their own special P-code operation.

[0129] Figures 20 and 21 illustrate the operations which are performed when a travelling program transmits itself to a predetermined recipient. In block 398, any program authorizing information is first checked to insure that the traversal operation is permitted. (It is conceivable that some travelling programs may not be permitted to travel -- but simply to do some function which terminates at the first use). In the rare case that the program is not allowed to travel, a special return code is presented to the caller.

[0130] The present embodiment implements the "TRAVERSE" operation as a built-in function. Furthermore, the function is defined to return "0" to the immediate caller of the function and "1" to the caller after the function is restarted on the recipient's computer. As explained earlier, this difference in return code allows the program to differentiate between the sender's and recipient's computer.

[0131] To do this, in block 399, the TRAVERSE function first pre-loads the value "1" on the execution stack, knowing that the stack is transmitted intact. This is the value that will therefore be returned when the travelling program is reconstituted and restarted on the recipient's computer. Then all relevant variable data such as the "variable" information table, process control blocks, the various stacks, variable control blocks are collected into a transmission format such as a format shown in Figure 2.

[0132] As indicated at block 402, the travelling program header is constructed and transmitted. The travelling program is transmitted segment by segment, although it could, in fact, be transmitted in a field by field format, or any other way if desired. Preferably, a hash is taken of each segment as it is transmitted.

[0133] Thereafter, in 404, the program and any authorizing information from the input file received with the travelling program is then copied to the output transmission file. The "variables" segment is then transmitted including the name, current value, and relevant status of each variable (406) Any certificates which were collected as part of performing digital (authorizing) signatures during this or previous traversals are then transmitted. Thus, any time a digital signature operation is performed, all the associated certificates are collected and transmitted in the certificate section of the travelling program 408. The signatures are maintained as variables within the program (i.e., within variable control blocks). Certificates in the presently preferred embodiment are treated as material which can be accessed via built in function calls.

[0134] Alternatively, it would be possible to include in the certificate package even those certificates which relate to the signatures of the overall transmission and signature(s) which authenticate and authorize the program itself. How-

ever, this would require that all the certificates definitely be known at the time the Certificate segment was written, and the logic, and possibly the position of the segments would need to be re-ordered to insure optimized processing.

[0135] In our implementation, we prefer to keep the certificates associated with the program's authorizing signature with the program authorization information in the header or program segment, and the certificates for the user-to-user transmission signature authentication with the signature in the closure segment.

[0136] After the certificates are transmitted, all file control blocks are examined resulting in the examination of all files which may have been transmitted during prior traversals and any newly attached files 410. A check is then made in block 412 to determine whether there are any more file control blocks to examine. A check is then made at block 414 to determine whether any file being examined was scheduled to be detached 414. If so, the routine branches back to 412 and neither the file, nor the file tag is copied for transmission. If the file is not scheduled to be detached, then the file tag name is copied into the transmission 416.

[0137] A check is then made to determine whether the file in question is part of an incoming travelling program which is being carried forward (418). If it is determined that it was part of the incoming traversal, then all file attributes from the incoming traversal as well as the file itself is copied to the outbound transmission file (422). This input file name may be accessed via the execution control area XCA and the input position of the file is associated with the file control block 422.

[0138] If the file is not part of an incoming traversal but rather was attached during the travelling program execution, then the file, the file type, and its attributes are copied into the transmission file 420. Thereafter, the routine branches back to block 412 to determine whether there are any more file control blocks to examine until all file control blocks have been examined.

[0139] As indicated in Figure 21, when all FCB's have been examined, a check is made to determine whether an overall user-to-user digital signature has been requested is required by the system program 430. Such an overall signature would be useful in detecting tampering with transmitted information.

[0140] If an overall digital signature is to be taken, then a digital signature operation on the hash of all material transmitted is performed (432). The digital signature operation may be performed in accordance with the teachings of U.S. Patent 5,005,200 (or more conventional digital signature techniques which do not have the associated authority verification attributes, as desired). As indicated at block 432, a hash was previously taken for each part of the transmission. It is noted that alternatively, a hash may be taken of each of the hashes. The digital signature step may involve user interaction to perform the signature.

[0141] Thereafter, validation is supplied at the end of transmission as the "closure" segment. The validation is supplied by transmitting a hash reflecting prior material. The signed hash should demonstrate user-to-user authentication 434. Any certificate necessary to validate the final signature, which are not already in the certificate segment, should be included in the CLOSURE segment. Thereafter, the transmission is closed 436.

[0142] Finally, in block 437, the value "1" which was previously loaded onto the execution stack for the benefit of the transmitted program when it arrives at the recipient, is removed and replaced with the value "0" -- which is returned to the current caller to allow it to distinguish itself.

[0143] Because creating a digital signature typically involves user interaction -- such as possibly selecting a certificate and opening the private key, or asking the user to operate his digital signature token device -- the material described in Figure 20 and 21 will actually operate in the preferred embodiment as P-code interrupt routines. As an example, the TRAVERSE function code would trigger a P-code interrupt, in which the logic from blocks 399 to 430 would operate as a PRE-ROLLOUT routine, while the block at 432 might operate as a INTER-ROLLOUT routine since it may require the aforementioned user interaction. The blocks thereafter (434, etc) would operate as a POST-WAIT routine.

[0144] The travelling program can be designed as desired to transmit itself numerous time during its execution to various recipients. In such multiple transmissions, the variables can be changed prior to each transmission as appropriate. In this fashion, the program in the position to do processing distinct for each recipient in a manner which is implementation dependent.

[0145] Figure 22 illustrates a sequence of operations for attaching a file to the travelling program. The attach file routine responds to an identified file tag and an identified file name. As indicated at block 440, a check is made to determine whether a file control block with the same tag exists. If so, then the previous file with the same tag is deleted 442.

[0146] Thereafter, a check is made to determine whether the specified file name reflects an existing file which is accessible by the user. In this regard, the travelling program may be associated with program authorization information which defines the range of operations which the program is able to perform, including the ability to access files. Such program authorization information will be checked to determine whether the file name is accessible. If the file name is not accessible by the user, then an error code/message is returned to the user 446.

[0147] If the file name is accessible to the user, then a file control block (FCB) is built with the specified tag and file name and the file will be attached during the next and subsequent transmission of the travelling program 448. The routine is thereafter resumed with an indication that the file has been attached successfully.

[0148] Figure 23 illustrates how a file is erased from the user system. When an "erase" function is attempted to be executed, security codes are checked to determine whether the program is authorized to perform such an operation (450). If the security codes indicate that the program is authorized to erase the specified file (452), then an erase operation is performed and the routine branches back with an indication whether the file was successfully erased 454. Alternatively, if the program is not authorized to perform an erase operation, then the calling routine is returned with an error message indicating that the file could not be erased (456).

[0149] Figure 24 illustrates the sequence of operations performed in detaching a file from a travelling program. As indicated in block 458, a check is made to determine whether a file control block exists for the identified tag associated with the file to be detached. If no FCB exists, then the main routine is returned to with an error message indicating that the file could not be detached 462. If the file control block does exist as determined at 458, then the file control block is deleted at 460 and the main routine is returned to with an indication that the file has been successfully detached.

[0150] Figure 25 delineates the sequence of operations performed when a file is to be "exported", i.e., transformed into a user file. A travelling program may take a specified file, for example, representing a spreadsheet and convert such a file to a recipient user's file that remains with the user even after the travelling program has been sent to a further destination. The file to be "exported" will be identified by a tag and an output file name and, if desired, a rewrite indicator identifying whether the file may be rewritten.

[0151] A check is initially made as to whether a file control block exists for the specified tag 498. If no FCB exists, then an appropriate error indicating code is generated and the calling routine is returned to (504). If a FCB does exist with the specified tag, a check is made to determine whether the file is part of an incoming travelling program 500. If the file to be exported was not part of an incoming traversal, then it must have been attached by the user and already be present in the user's file and, accordingly an error message is generated indicating that one is not allowed to export a newly attached file 502. If the file was part of the incoming traversal, then a check is made to determine whether the specified file already exists (480). If so, then a check is made at block 482 to determine whether it is okay to rewrite the specified file. The check includes determining whether the program is allowed to modify the specified existing file (if no "overwriting"), or to erase and create the specified file (if "overwriting" is permitted). If not, then the block 484 is used to return an access error to the program. If the check at 482 indicates that it is okay to rewrite, a determination is made as to whether the file should be overwritten or whether new material should be added to the end of the file (486). If overwriting is indicated at 486, then the existing file is erased (488). A new file is created, if permitted by program authorizing security information and preparations are made to start writing at the beginning of the file (490).

[0152] If overwriting is not indicated at 486, but new material data is to be added at the end, then preparations to start adding at the end of the existing file are made, as indicated at block 492. Thereafter, the data is copied from the correct position at the incoming traversal file to the output file (494) and the main routine is re-entered with an indication that the exporting operation has been successfully performed (496).

[0153] Figure 26 illustrates an exemplary sequence of operation performed when material is to be digitally signed. In implementing the digital signature function, initially a check is made to determine whether a digital signing operation is permitted by the program as indicated at block 510. Whether a program is permitted to perform a digital signature operation will be controlled by program authorization information which is associated with the program and which is monitored every time the program is executed to ensure that unauthorized operations are not performed. If the digital signature operation is not permitted, then an error message will be generated rejecting the digital signature function call 511.

[0154] If the digital signature operation is permitted, then in block 514, the SIGN function prepares for user interaction by moving an image of the data to be signed, together with any parameters (such as any required authorization for the data content) to temporary storage in preparation for receipt by the INTER-ROLLOUT routine (shown in Figure 27) which will perform the user interactions associated with performing the actual signature.

[0155] In block 512, the P-code routine is signalled, with interrupt routines which are described below.

[0156] If the digital signature authorization is authorized, then a display panel must be presented to the user to solicit which certificate is to be used for the signature operation. The signature operation is preferably performed in accordance with the inventor's U.S. patent No. 5,005,200 which patent has been expressly incorporated herein by reference. The user may possess a wide range of certificates for performing digital signature operations including those constructed along the lines of U.S. Patent No. 5,005,200. The INTER-ROLLOUT routine is given control at block 509 after much of the storage is rolled out (the signature routine itself must remain in storage, of course).

[0157] If there are no certificates suitable for performing the signature, then control passes to block 515 which generates an error indicator to be returned to the sign operation. If there is only one certificate suitable for performing the signature, then it is automatically passed to (513). If there are more than one suitable certificates, then the user is asked to select (516). If the user declines (517), then this an appropriate error indicator is generated, and passed to the program (515). Otherwise, the chosen suitable certificate is passed to (513).

[0158] The associated private key is then located (513) If block 518 determines that it is located on the user's token, then step (524) is used to solicit communication to the token so that it can perform the digital signature. Otherwise, the

user's private key is located in the system encrypted under a secret password phrase. The user is solicited (520) for this password, which is used to decrypt the private key. Any errors or bad passwords are detected, an appropriate error message is generated. To inhibit guessing by someone other than the true user, only a limited number of tries to give the correct password are allowed.

[0159] In block 522, the password is used to decrypt the private key, which in turn is used to sign the message, according to the necessary authority. After the operation, all traces of the secret material is erased, and the signature and certificate are returned to (268, Fig. 15) in temporary storage. In (270) control is then given to the POST-WAIT routine (530) which moves the signature from temporary storage to the execution stack.

[0160] In block 532, the operation is checked, and if it was successful, the proof hierarchy for the signer's certificate is obtained. Certificates are added to the overall certificate collection (maintained in the XCA (90, et al)) if they do not already appear.

[0161] Figure 28 illustrates the sequence of operations performed when displaying information to the user. The travelling program has associated therewith a display layout capability which is described in conjunction with Figure 28. The layout capabilities of the travelling program adapt functions heretofore associated with typesetting applications for use in a user interactive display mode together with additional enhanced capabilities.

[0162] The screen may be laid out such that input fields can be readily moved and associated with various attributes for very flexibly interacting with the user. Various display related operations and functions are summarized in block 540. The display presents an output based on a specified layout definition process controlled by the display processing portion of the interpreter.

[0163] The display processing involves analyzing conditional attributes and static attributes for the fields and the group of fields in the layout definition. In the display processing subroutine, variable substitution and iteration using conditional logic is performed as necessary. Although variable substitution is permitted, the system retains association between an input variable and where the field is to be displayed on the screen in the corresponding variable control block (VCB) even if the field is flowed into its final output position as dictated by the layout definition.

[0164] The following attributes are then provided to each field including, color, font, boldface/italics, style, size, underlining, blinking, reverse video, non-display (e.g., for hiding passwords), high intensity display, etc. Additionally, possible error messages are inserted where appropriate for a detected error condition and the proper cursor position is indicated.

[0165] The layout language used in block 540 permits not only the definition of a screen output but also definitions for accepting input. As indicated in block 542, fields are written to the user's terminal allowing input fields, as appropriate depending upon the application. As previously described, data structures may be rolled out to auxiliary storage (544) and rolled back (546) after the user performs data entry into the appropriate input fields.

[0166] To do this, the step 544 actually involves signaling a P-code interrupt, and having the block 545 executed as the associated INTER-ROLLOUT routine, and block 546 executed as the POST-WAIT routine responsible for mapping the input fields back to the VCBs for the associated variables. This may involve passing data through temporary storage.

[0167] Thereafter, the input is analyzed and the input data is inserted in all associated variables. A field validation is then performed for all input fields 548. Thus, a check may be made to make sure that for numeric fields only numbers have been entered. Similarly, a check may be made to determine whether an input field has the specified attributes.

[0168] Thereafter, a check is made at block 550 to determine whether there has been an error in any field. If there has been an error, then an error message is produced and the cursor is positioned to the errant field (552), after which the routine branches back to 540 to generate an error message display.

[0169] If the check at 550 fails to reveal an error in a particular field, then a further check is performed to cross verify that the fields are correct in context (e.g., although two adjacent fields may be correct individually, an error condition may be defined regarding the combination of fields) 554. Based on a cross verification, a determination is made as to whether the field contains an contextual error. If not, then a return is made to the caller 558. If there is a contextual error then an error, message is produced in accordance with block 552.

[0170] It should be noted that verification of both the individual fields is completely under control of the program. There may be various specifications, utility routines and other conveniences to simplifying handling common situations, but in general, any possible validation is possible. Cross-validation of fields may involve more semantic concerns, and is thus more likely to require specialized programming.

[0171] Figure 29 delineates a sequence of operation performed by a time delay routine. The time delay function may be utilized to wake up at predetermined time intervals and check to see whether any incoming electronic mail has arrived and attach itself to that mail to thereby efficiently handle incoming electronic data interchange. Thus, though such a time delay mechanism, a travelling program could examine a particular mail box at predetermined time intervals to check whether any mail has arrived. If the mail has arrived, the travelling program could send the mail to a destination to be handled by a further recipient. Alternatively, the travelling program could examine incoming data (such as mail), and based on various content indicators, automatically perform a traverse and spawn a new "instance" of itself which could treat the mail appropriately. Of course, the original "instance" could continue executing and process every in-

stance that arrives.

[0172] For example, if the incoming information happened to be EDI transactions, then a travelling program could read the information (using, for example, a READ built-in function), break it apart into internal variables, determine by whom it should be processed, and perform the appropriate traversal. Once successfully routed, the letter could be disposed, moved or archived, the program could clear its variables, and resume looking for more input.

[0173] Alternatively, after determining the type of material arrived, it could invoke another program to process the incoming data. If the other program happened to be a travelling program, then that program could be given the necessary input information, and could then TRAVERSE itself appropriate to the handling.

[0174] This would allow, for example, one travelling program to act as a automatic router for incoming data, such as EDI transactions, and then hand off to other travelling programs the transactions which it is not prepared to handle itself.

[0175] Furthermore, if the EDI were signed, then the travelling program could verify the signature immediately. If the signature were valid, and especially if it were done according to U.S. Patent No. 5,005,200, then the authorization for the content could be programmatically screened, and the travelling program could automatically spin-off an instance to handle the incoming transaction.

[0176] For example, with proper enhanced authorization, an incoming Purchase Order could be automatically and instantly routed to the shipping department to commence filling.

[0177] Items which arrived which were not signed, or which used simple signatures rather than authorizing signatures, could be routed to various clerical persons for exception processing and more detailed inspection.

[0178] As indicated in block 570, the time delay routine, sets the system alarm clock for a specified time. Thereafter, an optional roll out of data to auxiliary storage may be performed (572) by scheduling a P-code interrupt with appropriate routines followed by a performance of a roll-in of data after the specified time period has elapsed. Thereafter, a return to the calling routine occurs (576).

[0179] Figure 30 which shows the sequence of operations for a "select from directory" function. The directory could be a directory of files or a directory of user's, etc. Initially, a list is created of all candidate items 580. Thereafter, a display is generated to display at least part of the list 582. The user will have an opportunity to select among those items presented (583, 585), after which the function will return the names of the selected items, either as a function result or a set of special variables (584).

[0180] Again, as described elsewhere, the actual WAIT is performed through the use of the P-code interrupt function. In this case the INTER-ROLLOUT routine waits for the user to select from the selection, and returns the input to the program variables through the POST-WAIT routine.

[0181] Figure 31 is a routine which demonstrates how the interpreter program permits a user to perform digital signatures. As indicated at block 600, the data to be digitally signed is assembled based on data which the program is able to access: this includes user supplied input, data read from files, data accumulated from previous traversals, data based on the user's environment (e.g., the user's TSO identifier), the time, data incorporated into the program itself, and data derived from built-in functions (such as the built-in X12 data dictionary). Appropriate information is displayed to the user (602). The user then decides whether he or she wishes to sign the data, as indicated at block 604. If the user indicates he wishes to perform the signature, the system invokes the sign function, as illustrated in Figure 26, to further interact with the user and complete the signature (606). Thereafter, the digital signature is generated and saved as a program variable 608.

[0182] Figure 31 and the flowcharts which follow depict, in part, how a user might utilize the travelling program methodology described therein, while performing relatively few operations to accomplish powerful functions built into the aforedescribed interpreter.

[0183] Figure 32 exemplifies how a user would verify received information. As indicated in block 610, the data which is expected to be verified are assembled. Thereafter, a "verify" function with the assembled data and the saved digital signature, together with any possible authority requirements is invoked. The verification function may be accomplished as described in U.S. Patent No. 5,005,200 or using standard digital signature techniques if a conventional digital signature operation was utilized to sign the variables. Thereafter, a determination is made based on the processing in block circuit 12 as to whether the signature is verified (614). If so, then the program execution continues. If not, an error condition results indicating that the data has been tampered with or that there has been some kind of programming error 616. Return codes are defined to allow the program to distinguish whether the signature was invalid, whether it supported authorization capability, and if so, whether the authorization was confirmed.

[0184] Figure 33 illustrates how a travelling program collects a file to be transferred. Initially, the program determines the file to be transferred by, for example, displaying to the user, a list of files 620. A check may be made to determine whether it is necessary to have user interaction in order to determine the file (622). If yes, then the user is prompted to determine the file to be transferred 624. If it is not necessary to have user interaction to determine the file, then the entire file contents are attached to the set of data to be transferred 626. The operation is accomplished using the attached functions set forth in Figure 22 which involves building a file control block as previously described.

[0185] Figure 34 illustrates the travelling program operations performed in reading data from a specified file. Initially

the file is determined containing the data to be read (630). Thereafter, data is read from the specified file and saved as program variables 632. Figure 35 illustrates how the travelling program may update or create a file from program variables. As indicated in block 640, the user file into which data is to be written is first determined. Thereafter, a function is invoked that writes program variables into the user file 642.

[0186] It should be understood, even if not explicitly described in every case, that any program function which could lead to data loss, alteration, damage or disclosure is subject to security controls. Such controls can be applied at the program level, and either be tied to the incoming program and possibly by combined in some predetermined fashion with those also imposed by the user.

[0187] Therefore, for example, in the above case, the travelling program could only read or write user's data files if the program were so authorized.

[0188] Security constraints exist for at least the following classes of functions:

Display data to the user.
Soliciting input from the user.
Performing digital signatures.
Reading data from user files.
Creating user files.
Erasing user files.
Writing data into user files.
Remaining user files.
Attaching user files.
Exporting attached files into user files.
Invoking an digital notary device.
Receiving incoming electronic mail
Reading the contents of electronic mail
Moving or archiving incoming mail
Deleting incoming mail.
Generating outbound electronic mail, or doing various types of data transmissions
Being coupled to various types of equipment, device and services (FAX, printers, office equipment, robot devices, manufacturing equipment, etc.)
Performing a program traversal.
Invoking external programs.
Accessing, updating, activating, erasing, altering, invoking, or attaching other travelling programs

[0189] Figure 36 is illustrates how a travelling program may be designed to be split and sent to a number of different recipients and Figure 37 demonstrates how the previously split programs may be merged.

[0190] Turning first to Figure 36, the travelling program may need to be split in order, for example, to acquire survey data from a number of different recipients or to collect or distribute data to a number of different executives in an organization. Initially, the travelling program performs various housekeeping operations to prepare to split 650. Thereafter, variables are set in accordance with particular application requirements, e.g., the survey run by a particular user 652. Destination users are then determined and the traverse function is invoked as per Figures 20 and 21 to transmit the image of the programs, the programs variables together with any other appropriate data tailored to the individual recipients 654. The transmitted variables may change from instance 1 (656) to instance 2 (658), instance 3 (660), to instance N (662).

[0191] A check is ultimately made to determine whether there are more destinations to which to transmit (664). If so, then the routine branches back to 652 to transmit to the further destination. If there are no further destinations, then the final transfer is performed 666 in the same manner as explained above with respect to 654 to result in the final "instance" 668, thereafter resulting in the completion of the splitting operation.

[0192] In other examples, it may also be that the master program simply goes into some other processing. Perhaps, if it were running in a batch environment as an input distributor, and all the input were presently exhausted (having just been spun off to a number of users), it would go into a delay until something else arrived.

[0193] Turning to the Figure 37 merge operation, the travelling program has the intelligence to transfer itself from user to user to merge further data until the merging operation is complete. Initially, the travelling program arrives at a merging destination and is executed (680). A check is made to determine whether this is a master "instance" which is determined by a predetermined variable being set. If it is determined that this is not a master instance at 682, a slave instance is identified 684. At (685) the slave program checks if it has been invoked with the special "DEBRIEF" parameter (which is simply a convention used by this program ot determine when the slave is being called by the master), and if so (687) passes back all pertinent information to the master instance, then exits. If this is not the DEBRIEF

invocation, then a check is made to determine whether the master instance is available, i.e., has already arrived, 686. If the master instance is available then a call is made to the master instance 696, through the use of the call shown in Figure 18. After the master instance has been invoked, the routine branches back to block 680. If the master is not available, a message is issued that the master control for the series has not arrived 688.

[0194] Presuming the master instance has arrived and has been invoked, then at block 682 a determination is made that this is the master instance and a check will be made to determine whether any other slave instances have arrived 692. If so, then the slave instance will be invoked with a predetermined parameter to initiate the collection of data (referred to perhaps as "debriefing") 694. At entry point E, data is collected from the instance and is returned to the master and is written to a collection file 706. Thereafter, the instance that has just been invoked is erased 708 and the routine branches back to 692 in which case further information is collected if other instances have arrived.

[0195] If no further instances have arrived the file is checked to see if all instances have all arrived (698). If they have, as determined at 700, then the data could be read from the collection into variables in the travelling program. Depending on the expected size of the collection file, and the nature of the processing, it might be more desirable for the master program to process the completed file at that moment and either traverse itself to the next destination. or to encapsulate the result into a simple message, perhaps even an EDI transaction and simply transmit that raw data.

[0196] In other cases it might be appropriate for the program to ATTACH the file to itself and transfer it wholesale to another process. The file is erased and aggregate data is transmitted to the next destination 704. If all instances have not yet arrived, then a message is issued such as "waiting for forms to arrive" (702) and the routine is temporarily existed.

[0197] Figure 38 shows an alternative approach to merging previously split travelling program information. As shown in block 710, the travelling program arrives at a merging destination and is run. The collected data is then written to a special file 712. A check is made to determine whether all other instances have arrived as indicated at 714. If so, then the collected data is processed 716. and the program traverses to the next destination 718 and the routine is exited. If all other instances have not arrived as determined at 714, then a message is displayed such as "waiting for more forms to arrive" (720) and the current instance is deleted 722, and the routine is exited.

[0198] Figure 39 is a flowchart indicating how the travelling program has been designed to accommodate electronic data interchange (EDI) generation functions. Figure 39 more specifically demonstrates how a particular "X12" standard characteristic may be used. The X12 standard has an associated data dictionary and segment dictionary. The X12 segment dictionary, for example, may be used to define all segments necessary to define a purchase order. Each segment is defined as being a piece of data which is then looked up in a dictionary. Because there are many different ways to specify the quantity of an item, many variations of data are permitted in X12.

[0199] The present system embeds the X12 data dictionary into the interpreter which may be called as a built-in function. As indicated in block 720, initially a call is made to the X12 subroutine by specifying a segment name and items "XX, YY, WW,..". The program can provide X12 data code for popular common options typical in the organization's environment, so as to build a short list of options in order of normal usage. Examples of such items are, in a purchase order context, item number, part number and quantity. This call will result in a call to the built in data dictionary.

[0200] A check is made to determine whether the short list is empty (as indicated in 724). If so, the segment name is used to call the built-in function X12SEGLIST that locates the segment dictionary table of all associated data options 736. Thereafter, XI2DATANAME built-in function would be used to expand the data dictionary each associated description data 738 and the long complete list would be displayed 740.

[0201] If the check at 724 indicates that there is a short list, the XI2DATANAME data dictionary is used to locate the expanded description of each of the options on the short list. Thereafter, the short list is displayed 728. Then a check is made to determine whether the user wants the full long list as indicated at 730. If the answer is yes, then block 736 is executed as described above. If no, then the user's selection from either the short list or the long list is accepted (732).

[0202] A check is then made at block 734 to determine whether all data is collected. If so, we assemble and emit the completed X12 transaction 742 and then exit the routine. With respect to the emitting operation referred to in conjunction with 742, the present invention contemplates the capability of mailing specific sets of X12 data in addition to mailing the entire travelling program. If all data is not collected as indicated by the check in 734, then more data items are retrieved and the routine execution is repeated.

[0203] Figure 40 relates to the use of the travelling program in receiving an electronic data interchange transaction. For example, a particular user may have received a travelling program generated purchase order. Initially, the received EDI transaction is read 750. Perhaps by a timer-delay travelling program, as described with Figure 29, which spawns copies of itself as input arrives. The encoded EDI is then parsed into program variables 752. The received EDI is then moved to an archive repository to preserve that which has been received for possible audit. The segments are then processed via a coupled segment dictionary 756. The segment rules associated with X12 are enforced which, for example, may relate to not having certain kinds of data in particular fields, 758. For each data item, the data dictionary associated with each segment is located 760. For a statement such as shown in 762 where DESC=X12DATANAME (SEGCODE, DATA ITEM), this statement will result in a call to the data dictionary to get a meaningful description of the data item. The retrieved meaningful description will be placed into a display variable resulting in, for example, a

display of the purchase order in a purchase order format. All data items are processed by branching back to block 762 and all segments are processed branching back to 756.

[0204] The preferred embodiment also allows access to a Digital Notary facility by providing built-in functions which can access a digital notary, or notary device such as described in inventor's U.S. Patent No. 5,001,752 (which is incorporated herein by reference), or other devices as well.

[0205] By allowing a travelling program to access such a facility, the travelling program is able to move data to a platform where the digital notary can be easily accessed, then using the built-in function to do so. This allows notarization for important signatures, timestamps for inbound traffic, or for any other reason. Since such notarization is strictly under control of the program, any criteria whatever, whether automatic or based on user requests, can be used.

[0206] Also as described earlier, the facility allows for the coupling to outbound FAX so that electronic forms, in addition to being converted to EDI, or printed, can also be faxed to the ultimate recipient.

[0207] Also, as implied, but not explicitly stated, even when a travelling program emits an EDI transaction, it may still be activated later. One example would be a travelling program which first serves as an electronic requisition then, after sufficient approving signatures, generates a purchase order. It could then send itself to a repository where it could later be reactivated when the corresponding invoice and bills eventually arrive (electronic or otherwise) and can serve as a method for reconciling the order with the shipment received and the billing. It can incorporate logic which to keep track of which items have been received, and which are still pending. Because of the ability to flexibly direct itself, it can span many different sites. Insofar as handling shipping and receiving, it is also possible to couple the travelling program with a bar code reader and validate materials sent and received without human data entry.

[0208] The preferred embodiment envisions that the travelling program could be coupled to a variety of equipment, including office equipment, and other devices and facilitates.

[0209] Also, any given traversal could also be sent simultaneously to a variety of recipients.

[0210] The following listing reiterates and summarizes many of the above-described functions (and identifies some additional functions) which the preferred embodiment is capable of performing. This list is only illustrative and is not intended to be exhaustive of the many other applications to which the present invention may be advantageously applied.

Displaying data to the user using a layout language (similar to, e.g, TxX, or SCRIPT),
Soliciting input from the user using a layout-type language (similar to, e.g., TeX, or SCRIPT).
Performing digital signatures for data computed under program control.
Verifying digital signatures based on data computer under program control.
Handling co-signatures, possibly including routing suggestions derived from the signer's certificates.
Reading data from user files,
Creating user files,
Erasing user files
Writing data into user files
Renaming user files
Receiving incoming electronic mail
Reading the contents of electronic mail
Moving or archiving incoming mail
Deleting incoming mail
Generating outbound electronic mail.
Coupling to and controlling an outbound FAX server
Coupling to and controlling a printer.
Generating a graphical image.
Coupling to and controlling a device that can receive and transmit audio signals
Accessing various types of equipment, including office equipment, computer equipment (tapes, disks, etc.) robot devices, manufacturing equipment, etc.
Splitting an instance of the travelling program into several instances by virtue of multiple traversals.
Being able to re-combine the data contained in the several travelling programs, possibly not even reflecting the same program, into a single form.
Erasing other instances of travelling programs.
Invoking external programs.
Invoking other travelling programs as subroutines.
Activating other travelling programs as independently executing functions.
Extracting data from a dormant (non-executing) travelling program.
Determining information about another (non-executing) travelling program without have to execute it -- such as name of the program, and other status, etc.

Extracting information from the certificates associated with digital signatures. This information being used to

help direct routing if cosignature requirements are involved.

Making a copy of a travelling program as a data variable within another program, or ATTACHing a travelling program as a file to another.

Using one travelling program (the "carrier") to transport a new version of another to various destinations, and replacing the program segment of existing instances with another, more up-to-date version of the program. One way to do this would be for the newer program segment to be added to the end of existing travelling programs. Enhancements to the existing interpreter/loader would recognize that a program segment following the closure segment reflected a suggested program revision. After whatever normal transmission was performed, it would then validate the digital signatures associated with the proposed revised program, and, if they carried the proper authority, would then commence using the new program in place of the program which had arrived as part of the standard traversal.

Attaching user files.

Exporting attached files into user files.

Detaching previously attached files.

Accessing a digital notary device

Performing a program traversal

Transmitting user data (in other than a traversal), so that the transmission does not include the travelling program itself, (e.g., simply sending a message to another destination.

Using built-in functions to simplify the use, creation, display, construction and receipt of EDI (such as X12 or EDIFACT) to conveniently supply common information and facilities without having to supply these functions in the travelling program. This includes built-in functions which access the Data Element Dictionary, the Segment Dictionary, the segment rules, and the transaction sets themselves.

[0211] While the invention has been described in connection with what is presently considered to be the most practical embodiment, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the scope of the appended claims.

Claims

1. Method for processing information, said information consisting of digital instructions and accompanying data, among a plurality of computers (Terminals A, B ... N) coupled to a channel (12), over which computers exchange messages, said computers being part of a digital communications system, said method comprising the steps of: executing on a first computer a sequence of digital instructions (Fig. 2, block 22) including instructions which determine at least one next destination that receives the sequence of digital instructions together with the accompanying data; and transmitting said sequence of digital instructions together with the accompanying data to said next destination;
characterized in that
said accompanying data includes at least one digital signature (432) which is selectively applied to said information; and in that,
under the control of said sequence of digital instructions, a digital signature verification operation based upon said information is performed.

2. A method according to Claim 1, wherein said digital signature is represented as data subject to being logically processed by said sequence of digital instructions.

3. A method according to Claim 1 or Claim 2, further including the step of associating of a digital certificate with said digital signature and wherein said digital certificate is represented as data subject to being logically processed by said sequence of digital instructions.

4. A method according to any preceding claim, further including the step of acquiring data from a user at at least one of said plurality of computers, and translating the acquired data by said sequence of digital instructions into a predefined data structure conforming to a recognized standard.

5. A method according to Claim 4, including the step of processing and verifying the digital signature and the data to which it is applied.

6. A method according to any preceding claim, further including the step of translating data under direction of said sequence of digital instructions into an Electronic Data Interchange (EDI) format.

7. A method according to any preceding claim, including the step of logically constructing the information to which the digital signature can be selectively applied, wherein such information is treated as a program variable on which said sequence of digital instructions operate.

8. A method according to any preceding claim, further including the step of performing a digital signature operation as a function which is invoked under control of said sequence of digital instructions.

9. A method according to Claim 8, wherein the data supplied to said digital signature function reflects values based on any of a set of data read from a user's file, data built into said sequence of digital instructions, data entered by the user, data obtained from other signatures, and data obtained from a digital certificate (514).

10. A method according to any preceding claim, further including the step of including an indication of the authority which has been vested in a user performing a digital signature (432), by including sufficient digital information to allow verification that the authority exercised by the signer was properly exercised (434).

11. A method according to any preceding claim, further including the step of selecting from a collection of digital certificates the certificate to be used in performing a digital signature.

12. A method according to any preceding claim, wherein the digital signature is generated using a private key.

13. A method according to Claim 12, including the step of determining whether the private key is stored in a token device or a computer memory.

14. A method as claimed in any preceding claim, further comprising:

providing a sequence of digital instructions to control the management of digital signatures, including instructions which:
determine digital values,
control creation of a digital signature value based on determined digital values, and
determine said next destination;
executing in at least one digital, computer at least one of said instructions to determine a first digital value to be digitally signed;
executing in at least one digital computer at least one of said instructions to control the creation of a digital signature value computed on said first digital value;
executing in at least one digital computer at least one of said instructions to determine a next destination;
transmitting to said next destination digital information including said sequence of digital instructions;
executing in at least one of said computers at least one of said instructions to determine a further next destination; and
transmitting to said further next destination, digital information including said digital signature value.

15. A method according to Claim 14, wherein said information transmitted to said further next destination does not include said sequence of digital instructions.

Patentansprüche

1. Verfahren zur Handhabung von Informationen, welche aus digitalen Befehlen und zugehörigen Daten bestehen, unter einer Mehrzahl von Rechnern (Terminal A, B, N), welche mit einem Kanal (12) gekoppelt sind, über welchen Rechner Nachrichten austauschen, wobei die Rechner Teil eines digitalen Kommunikationssystems sind, wobei das Verfahren folgende Schritte aufweist:

Durchführen einer Folge von digitalen Befehlen (Fig. 2, Block 22) einschließlich Befehlen, welche mindestens einen nächsten Bestimmungsort festlegen, der die Folge digitaler Befehle zusammen mit den zugehörigen Daten empfängt, an einem ersten Rechner; und

Übertragen der genannten Folge von digitalen Befehlen zusammen mit den zugehörigen Daten zu dem genannten nächsten Bestimmungsort;

dadurch gekennzeichnet, daß

die genannten zugehörigen Daten mindestens eine digitale Signatur (432) enthalten, die selektiv an der genannten Information vorgenommen wird;

und daß unter der Steuerung der genannten Folge von digitalen Befehlen eine Prüfoperation der digitalen Signatur auf der Basis der genannten Information durchgeführt wird.

2. Verfahren nach Anspruch 1, bei welchem die genannte digitale Signatur als Daten dargestellt wird, welche durch die genannte Folge von digitalen Befehlen einer logischen Bearbeitung unterziehbar sind.

3. Verfahren nach Anspruch 1 oder Anspruch 2, welches weiter den Schritt der Zuordnung einer digitalen Beglaubigung zur digitalen Signatur umfaßt, wobei die digitale Beglaubigung durch Daten dargestellt ist, welche durch die genannte Folge digitaler Befehle einer logischen Verarbeitung unterziehbar sind.

4. Verfahren nach einem der vorhergehenden Ansprüche, welches weiter den Schritt der Annahme von Daten von einem Benutzer an mindestens einem der Mehrzahl von Rechnern und der Übersetzung der angenommenen Daten durch die genannte Folge von Befehlen in eine vorbestimmte Datenstruktur umfaßt, welche einem anerkannten Standard entspricht.

5. Verfahren nach Anspruch 4, welches den Schritt der Verarbeitung und der Prüfung der digitalen Signatur und der Daten umfaßt, an denen sie vorgenommen wurde.

6. Verfahren nach einem der vorhergehenden Ansprüche, welches weiter den Schritt der Übersetzung der Daten unter Leitung der genannten Folge digitaler Befehle in ein ELECTRONIC DATA INTERCHANGE-Format (EDI-Format) umfaßt.

7. Verfahren nach einem der vorhergehenden Ansprüche, welches den Schritt des logischen Aufbaus der Information umfaßt, an welcher die digitale Signatur selektiv vornehmbar ist, wobei diese Information als eine Programmvariable behandelt wird, auf welche die genannte Folge digitaler Befehle Einfluß nimmt.

8. Verfahren nach einem der vorhergehenden Ansprüche, welches weiter den Schritt der Durchführung einer Operation der digitalen Signatur als eine Funktion umfaßt, die unter Steuerung der genannten Folge von digitalen Befehlen angefordert wird.

9. Verfahren nach Anspruch 8, bei welchem die Daten, die der digitalen Signaturfunktion dargeboten werden, Werte repräsentieren, welche auf irgendeiner Gruppe von Daten, die aus einer Benutzerakte herausgelesen wurden, von Daten, die in die genannte Folge von digitalen Befehlen eingebaut sind, von Daten, die vom Benutzer eingegeben wurden, von Daten, welche von anderen Signaturen erhalten wurden, sowie von Daten, welche von einer digitalen Beglaubigung (514) erhalten wurden, basieren.

10. Verfahren nach irgendeinem der vorhergehenden Ansprüche, welches weiter den Schritt des Hinzunehmens einer Angabe der Vollmacht umfaßt, die einem Benutzer erteilt wurde, um eine digitale Signatur (432) durchführen zu dürfen, indem ausreichende digitale Information beigegeben wird, um prüfen zu können, daß die von dem Unterzeichner ausgeübte Vollmacht ordnungsgemäß ausgeübt (434) wird.

11. Verfahren nach einem der vorhergehenden Ansprüche, welches weiter den Schritt der Auswahl der bei Durchführung einer digitalen Signatur zu verwendenden Beglaubigung aus einer Sammlung von digitalen Beglaubigungen umfaßt.

12. Verfahren nach irgendeinem vorhergehenden Anspruch, bei welchem die digitale Signatur unter Verwendung eines privaten Schlüssels erzeugt wird.

13. Verfahren nach Anspruch 12, welches den Schritt der Feststellung umfaßt, ob der private Schlüssel in einem Beleggerät oder einem Speicher eines Rechners gespeichert ist.

14. Verfahren nach irgendeinem vorhergehenden Anspruch, welches weiter folgendes umfaßt:

Bereitstellen einer Folge von digitalen Befehlen zur Steuerung der Handhabung digitaler Signaturen, wobei die Folge digitaler Befehle ihrerseits Befehle enthält, welche:

digitale Werte bestimmen,

die Erzeugung eines digitalen Signaturwertes auf der Basis bestimmter digitaler Werte steuern, und

den nächsten Bestimmungsort festlegen;

Durchführen mindestens eines der genannten Befehle in mindestens einem digitalen Rechner zur Bestimmung eines ersten digitalen Wertes, der einer digitalen Signatur unterzogen werden soll;

Durchführen mindestens eines der genannten Befehle zur Steuerung eines digitalen Signaturwertes, der auf der Basis des ersten digitalen Wertes errechnet worden ist, in mindestens einem digitalen Rechner;

Durchführen mindestens eines der genannten Befehle zur Bestimmung des nächsten Bestimmungsortes in mindestens einem digitalen Rechner;

Übertragen digitaler Informationen unter Einschluß der genannten Folge digitaler Befehle zum nächsten Bestimmungsort;

Durchführen mindestens eines der genannten Befehle zur Bestimmung eines weiteren nächsten Bestimmungsortes in mindestens einem der genannten Rechner; und

Übertragen digitaler Informationen einschließlich des genannten digitalen Signaturwertes zu dem genannten weiteren nächsten Bestimmungsort.

15. Verfahren nach Anspruch 14, bei welchem die zu dem weiteren nächsten Bestimmungsort übertragene Information nicht die genannte Folge digitaler Befehle enthält.

Revendications

1. Méthode de traitement d'informations, les dites informations consistant en instructions numériques et en données d'accompagnement, parmi une pluralité d'ordinateurs (Terminaux A, B ... N) couplés à un canal (12), sur lequel les ordinateurs échangent des messages, les dits ordinateurs faisant partie d'un système de communication numérique, la dite méthode comprenant les étapes de :

exécution, sur un premier ordinateur, d'une séquence d'instructions numériques (figure 2, bloc 22) incluant des instructions qui déterminent au moins une destination suivante qui reçoit la séquence d'instructions numériques ainsi que les données d'accompagnement ; et
transmission de la dite séquence d'instructions numériques ainsi que des données d'accompagnement à la destination suivante ;

caractérisée en ce que :

les dites données d'accompagnement comprennent au moins une signature numérique (432) qui est sélectivement appliquée aux dites informations ; et
sous la commande de la dite séquence d'instructions numériques, une opération de vérification de signature numérique basée sur les dites informations est effectuée.

2. Méthode selon la revendication 1, dans laquelle la dite signature numérique est représentée sous forme de données soumises à un traitement logique par la dite séquence d'instructions numériques.

3. Méthode selon la revendication 1 ou la revendication 2, comprenant en outre l'étape d'association d'un certificat numérique à la dite signature numérique, et dans laquelle le dit certificat numérique est représenté sous la forme

de données soumises à un traitement logique par la dite séquence d'instructions numériques.

4. Méthode selon une quelconque des revendications précédentes, comprenant en outre l'étape d'acquisition de données d'un utilisateur à au moins un de la dite pluralité d'ordinateurs, et de traduction des données acquises, par la dite séquence d'instructions numériques, en une structure de données prédéfinie en conformité à un standard reconnu.

5. Méthode selon la revendication 4, incluant l'étape de traitement et de vérification de la signature numérique et des données auxquelles elle est appliquée.

6. Méthode selon une quelconque des revendications précédentes comprenant en outre l'étape de traduction des données, sous la direction de la dite séquence d'instructions numériques, en un format d'Echange de Données Electroniques (EDE).

7. Méthode selon une quelconque des revendications précédentes, comprenant l'étape de construction logique des informations auxquelles la signature numérique peut être sélectivement appliquée, dans laquelle ces informations sont traitées comme une variable de programme sur laquelle la dite séquence d'instructions numériques opère.

8. Méthode selon une quelconque des revendications précédentes, comprenant en outre l'étape d'exécution d'une opération de signature numérique comme une fonction qui est appelée sous la commande de la dite séquence d'instructions numériques.

9. Méthode selon la revendication 8, dans laquelle les données fournies à la dite fonction de signature numérique reflètent des valeurs basées sur un élément quelconque d'un ensemble de données extraites d'un fichier d'utilisateur, de données incorporées dans la dite séquence d'instructions numériques, de données entrées par l'utilisateur, de données obtenues à partir d'autres signatures, et de données obtenues à partir d'un certificat numérique (514).

10. Méthode selon une quelconque des revendications précédentes, comprenant en outre l'étape d'inclusion d'une indication de l'autorité qui a été dévolue à un utilisateur effectuant une signature numérique (432), par inclusion d'informations numériques suffisantes pour permettre la vérification de ce que l'autorité exercée par le signataire a été correctement exercée (434).

11. Méthode selon une quelconque des revendications précédentes, comprenant en outre l'étape de sélection, à partir d'une collection de certificats numériques, du certificat à utiliser dans l'exécution d'une signature numérique.

12. Méthode selon une quelconque des revendications précédentes, dans laquelle la signature numérique est engendrée au moyen d'une clé privée.

13. Méthode selon la revendication 12, comprenant l'étape de détermination de ce que la clé privée est stockée dans un dispositif à jeton ou dans une mémoire d'ordinateur.

14. Méthode selon une quelconque des revendications précédentes, comprenant en outre :

la préparation d'une séquence d'instructions numériques pour commander la gestion de signatures numériques, incluant des instructions qui :

- déterminent des valeurs numériques,
- commandent la création d'une valeur de signature numérique basée sur les valeurs numériques déterminées, et
- déterminent la dite destination suivante :

l'exécution dans au moins un ordinateur numérique d'au moins une des dites instructions pour déterminer une première valeur numérique à signer numériquement :
l'exécution dans au moins un ordinateur numérique d'au moins une des dites instructions pour commander la création d'une valeur de signature numérique calculée sur la dite première valeur numérique ;
l'exécution dans au moins un ordinateur numérique d'au moins une des dites instructions pour déterminer une destination suivante ;

la transmission à la dite destination suivante d'informations numériques incluant la dite séquence d'informations numériques ;
l'exécution dans au moins un des dits ordinateurs d'au moins une des dites instructions pour déterminer une autre destination suivante ; et
la transmission à la dite autre destination suivante d'informations numériques incluant la dite valeur de signature numérique.

15. Méthode selon la revendication 14, dans laquelle les dites informations transmises à la dite autre destination suivante n'incluent pas la dite séquence d'instructions numériques.

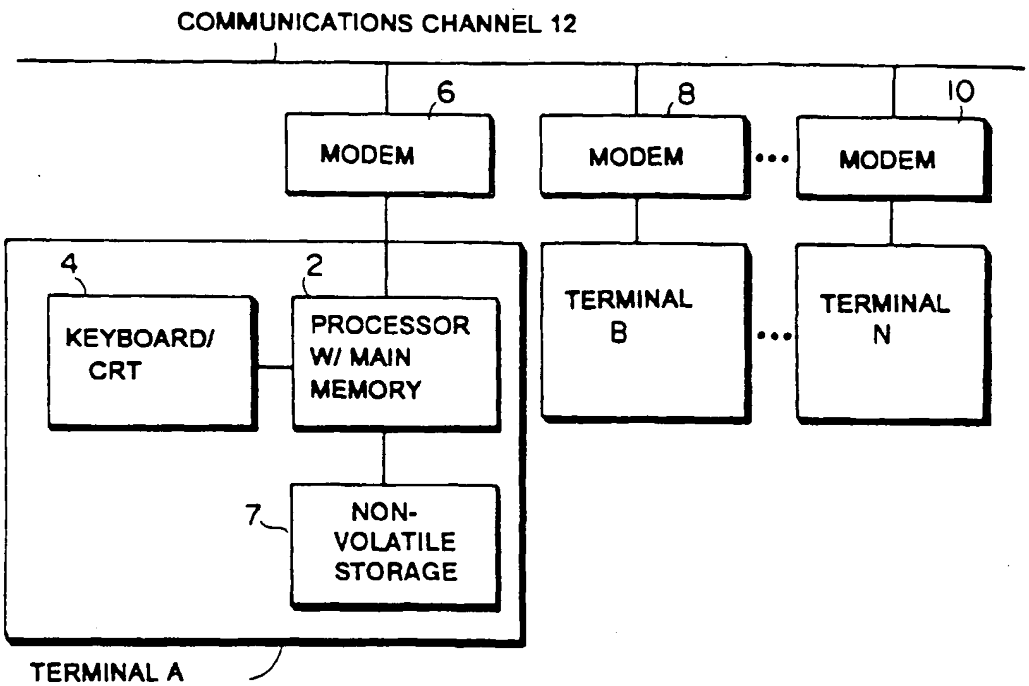

COMMUNICATIONS CHANNEL 12
6
8
10
MODEM
MODEM
MODEM
4
2
KEYBOARD/
CRT
PROCESSOR
W/ MAIN
MEMORY
TERMINAL
B
TERMINAL
N
7
NON-
VOLATILE
STORAGE
TERMINAL A

Fig. 1

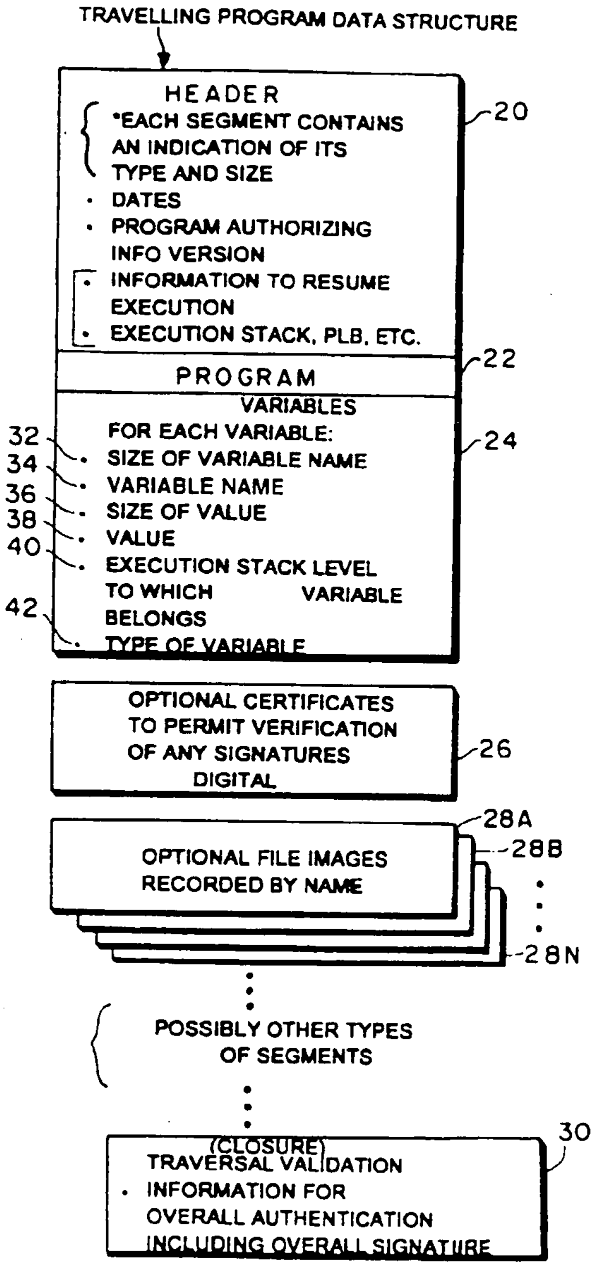
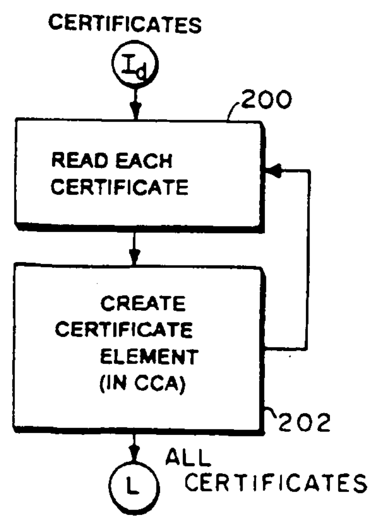

TRAVELLING PROGRAM DATA STRUCTURE
HEADER
•EACH SEGMENT CONTAINS AN INDICATION OF ITS TYPE AND SIZE
• DATES
• PROGRAM AUTHORIZING INFO VERSION
• INFORMATION TO RESUME EXECUTION
• EXECUTION STACK, PLB, ETC.
20
PROGRAM
22
VARIABLES
FOR EACH VARIABLE:
• SIZE OF VARIABLE NAME
• VARIABLE NAME
• SIZE OF VALUE
• VALUE
• EXECUTION STACK LEVEL TO WHICH VARIABLE BELONGS
• TYPE OF VARIABLE
24
32
34
36
38
40
42
OPTIONAL CERTIFICATES TO PERMIT VERIFICATION OF ANY SIGNATURES DIGITAL
26
OPTIONAL FILE IMAGES RECORDED BY NAME
28A
28B
28N
POSSIBLY OTHER TYPES OF SEGMENTS
(CLOSURE)
TRAVERSAL VALIDATION
• INFORMATION FOR OVERALL AUTHENTICATION INCLUDING OVERALL SIGNATURE
30

Fig. 2

Fig. 11


CERTIFICATES
I_d
READ EACH CERTIFICATE
200
CREATE CERTIFICATE ELEMENT (IN CCA)
202
ALL CERTIFICATES
L

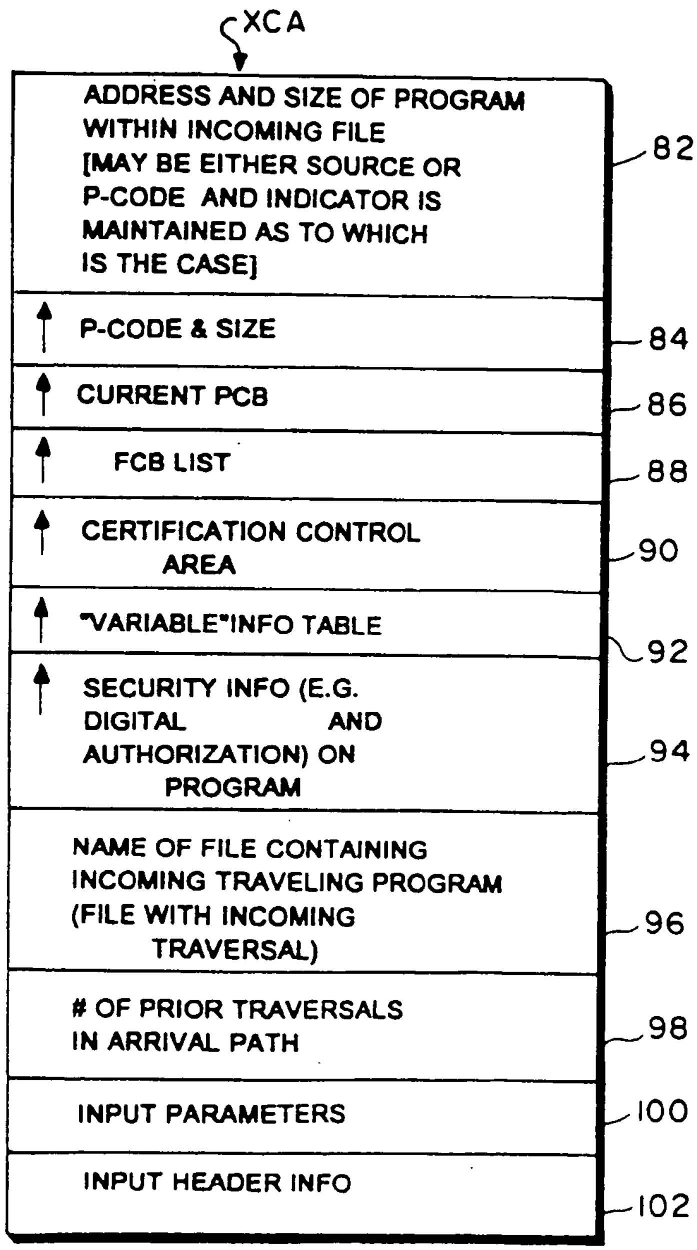

XCA
ADDRESS AND SIZE OF PROGRAM WITHIN INCOMING FILE [MAY BE EITHER SOURCE OR P-CODE AND INDICATOR IS MAINTAINED AS TO WHICH IS THE CASE]
82
P-CODE & SIZE
84
CURRENT PCB
86
FCB LIST
88
CERTIFICATION CONTROL AREA
90
"VARIABLE"INFO TABLE
92
SECURITY INFO (E.G. DIGITAL AND AUTHORIZATION) ON PROGRAM
94
NAME OF FILE CONTAINING INCOMING TRAVELING PROGRAM (FILE WITH INCOMING TRAVERSAL)
96
OF PRIOR TRAVERSALS IN ARRIVAL PATH
98
INPUT PARAMETERS
100
INPUT HEADER INFO
102

Fig. 3

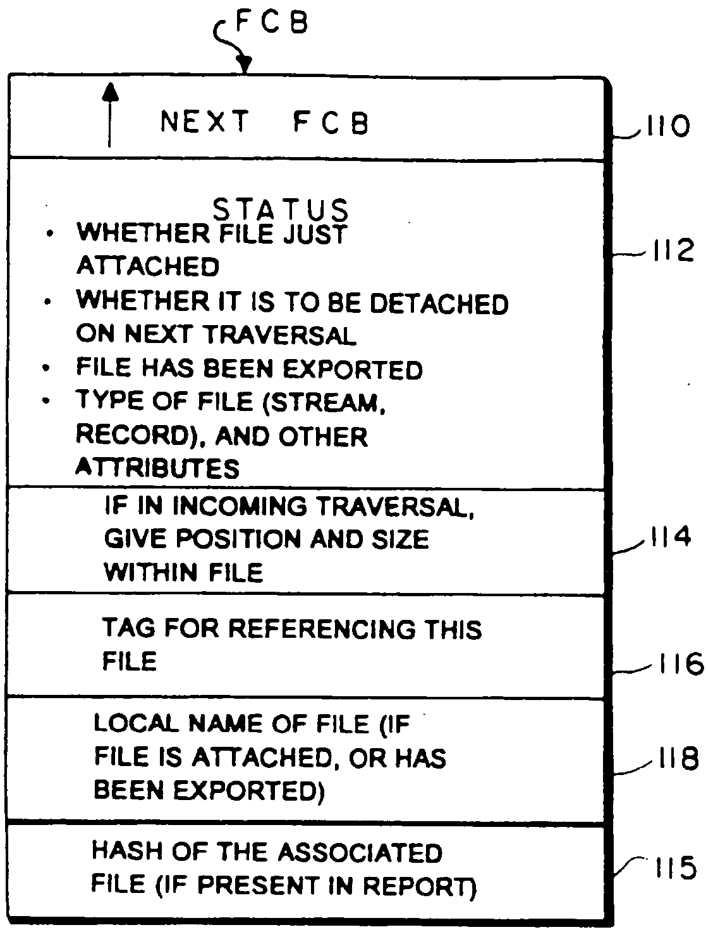

FCB
NEXT FCB
110
STATUS
• WHETHER FILE JUST ATTACHED
• WHETHER IT IS TO BE DETACHED ON NEXT TRAVERSAL
• FILE HAS BEEN EXPORTED
• TYPE OF FILE (STREAM, RECORD), AND OTHER ATTRIBUTES
112
IF IN INCOMING TRAVERSAL, GIVE POSITION AND SIZE WITHIN FILE
114
TAG FOR REFERENCING THIS FILE
116
LOCAL NAME OF FILE (IF FILE IS ATTACHED, OR HAS BEEN EXPORTED)
118
HASH OF THE ASSOCIATED FILE (IF PRESENT IN REPORT)
115

Fig. 4

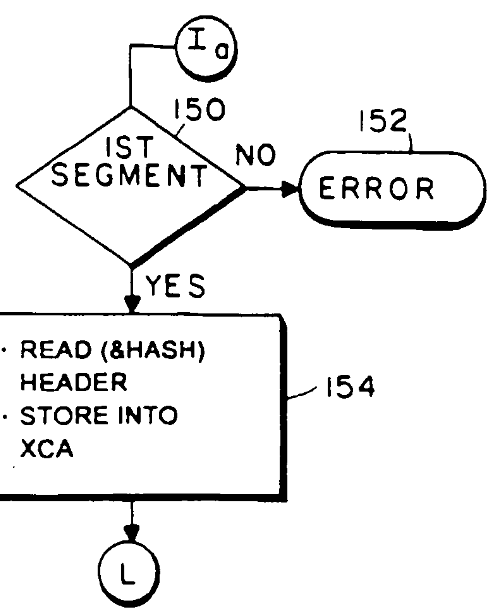

Ia
150
IST SEGMENT
NO
152
ERROR
YES
· READ (&HASH) HEADER
· STORE INTO XCA
154
L

Fig. 8

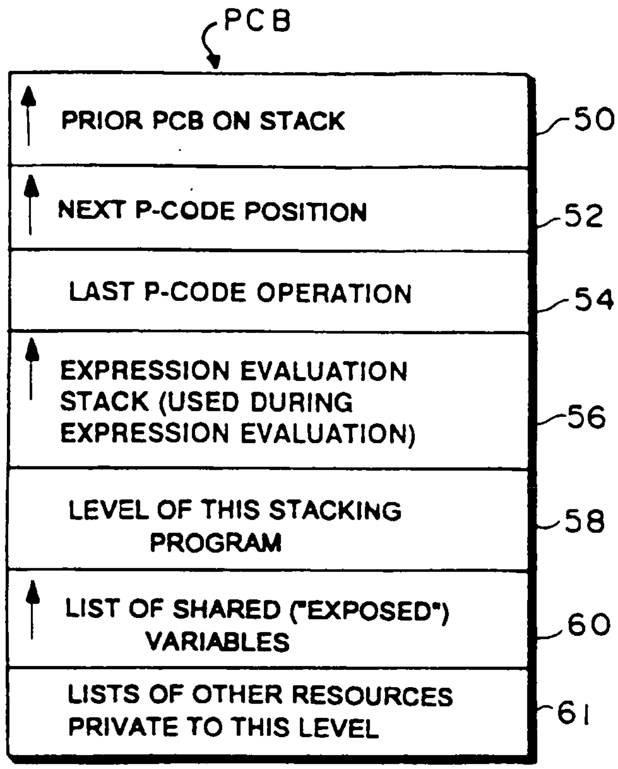

PCB
PRIOR PCB ON STACK
50
NEXT P-CODE POSITION
52
LAST P-CODE OPERATION
54
EXPRESSION EVALUATION STACK (USED DURING EXPRESSION EVALUATION)
56
LEVEL OF THIS STACKING PROGRAM
58
LIST OF SHARED ("EXPOSED") VARIABLES
60
LISTS OF OTHER RESOURCES PRIVATE TO THIS LEVEL
61

Fig. 5

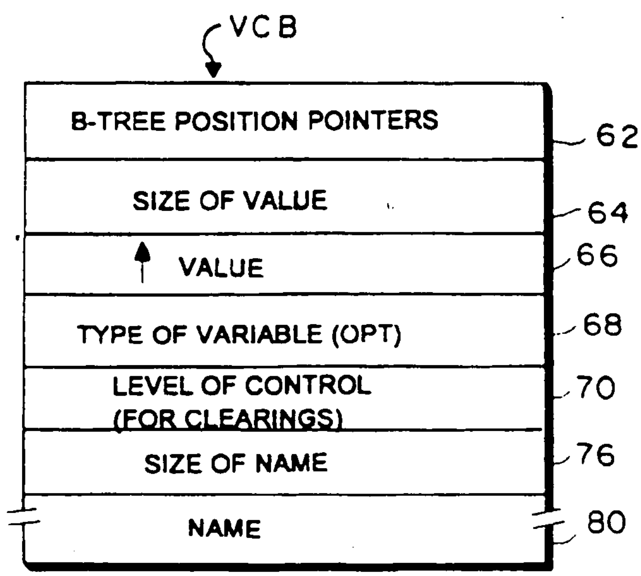

VCB
B-TREE POSITION POINTERS
62
SIZE OF VALUE
64
VALUE
66
TYPE OF VARIABLE (OPT)
68
LEVEL OF CONTROL (FOR CLEARINGS)
70
SIZE OF NAME
76
NAME
80

Fig. 6

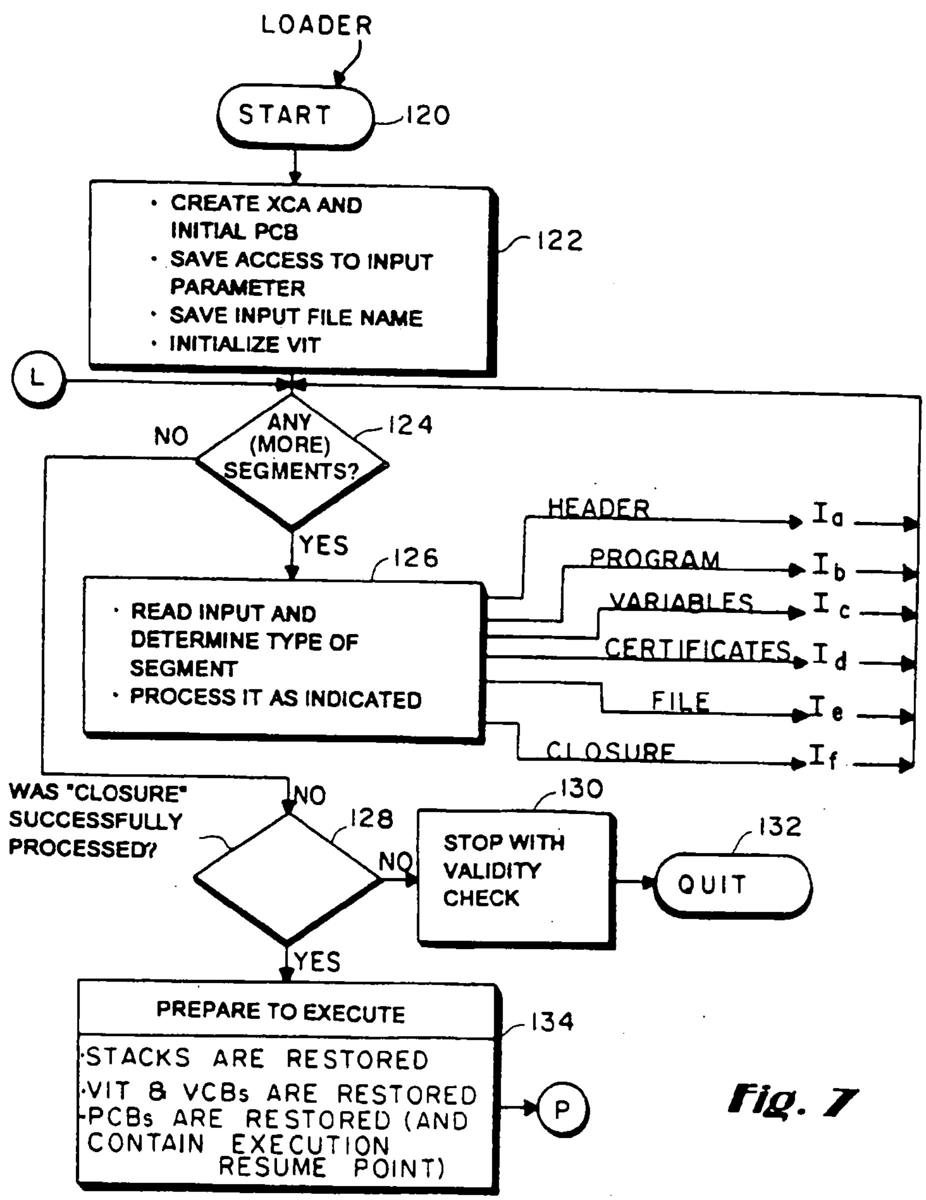

LOADER
START
120
• CREATE XCA AND INITIAL PCB
• SAVE ACCESS TO INPUT PARAMETER
• SAVE INPUT FILE NAME
• INITIALIZE VIT
122
L
ANY (MORE) SEGMENTS?
124
NO
YES
• READ INPUT AND DETERMINE TYPE OF SEGMENT
• PROCESS IT AS INDICATED
126
HEADER
PROGRAM
VARIABLES
CERTIFICATES
FILE
CLOSURE
Ia
Ib
Ic
Id
Ie
If
WAS "CLOSURE" SUCCESSFULLY PROCESSED?
128
NO
NO
YES
STOP WITH VALIDITY CHECK
130
QUIT
132
PREPARE TO EXECUTE
• STACKS ARE RESTORED
• VIT & VCBs ARE RESTORED
• PCBs ARE RESTORED (AND CONTAIN EXECUTION RESUME POINT)
134
P

Fig. 7

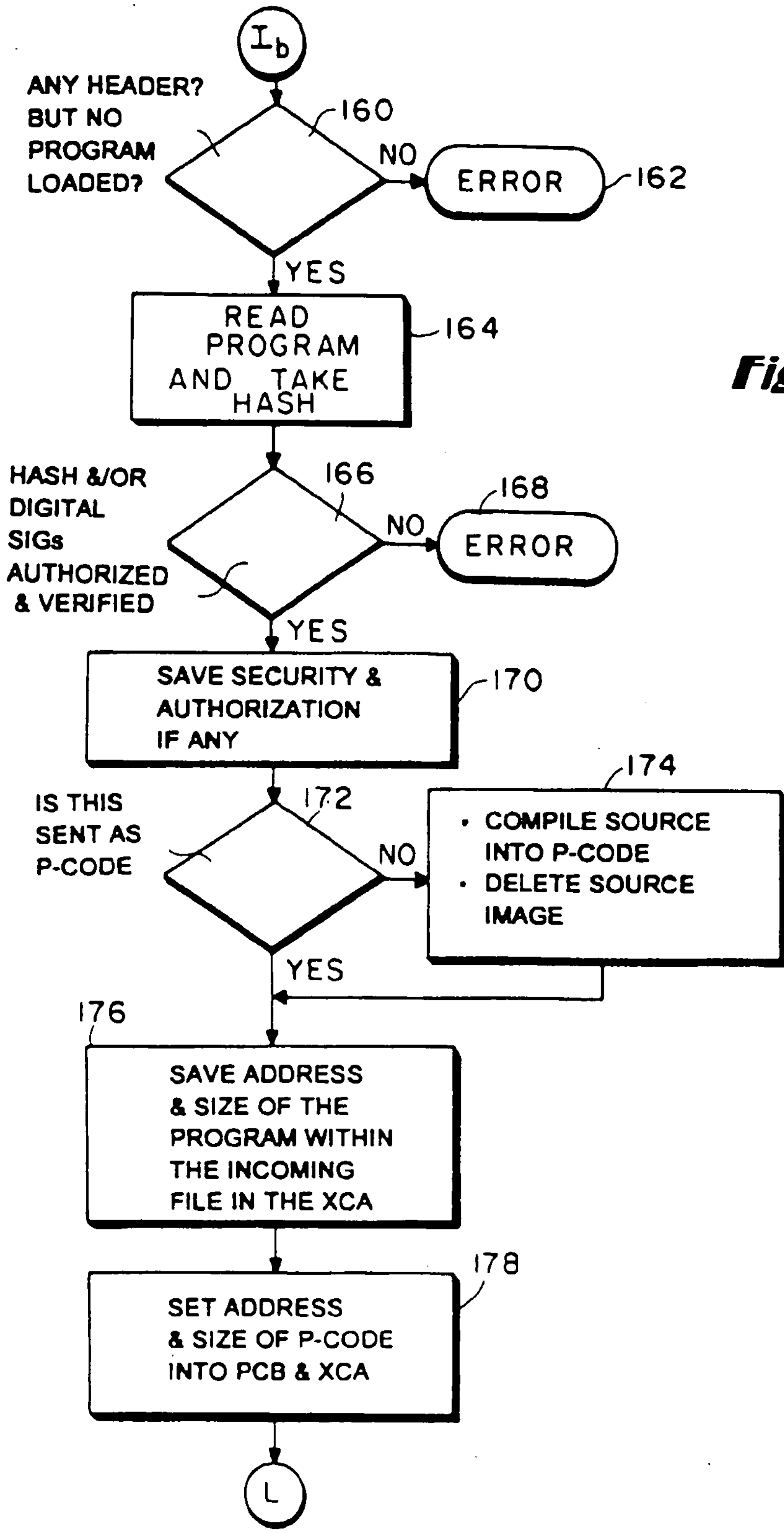

Ib
ANY HEADER? BUT NO PROGRAM LOADED?
160
NO
ERROR
162
YES
READ PROGRAM AND TAKE HASH
164
HASH &/OR DIGITAL SIGs AUTHORIZED & VERIFIED
166
NO
168
ERROR
YES
SAVE SECURITY & AUTHORIZATION IF ANY
170
IS THIS SENT AS P-CODE
172
NO
174
• COMPILE SOURCE INTO P-CODE
• DELETE SOURCE IMAGE
YES
176
SAVE ADDRESS & SIZE OF THE PROGRAM WITHIN THE INCOMING FILE IN THE XCA
178
SET ADDRESS & SIZE OF P-CODE INTO PCB & XCA
L

Fig. 9

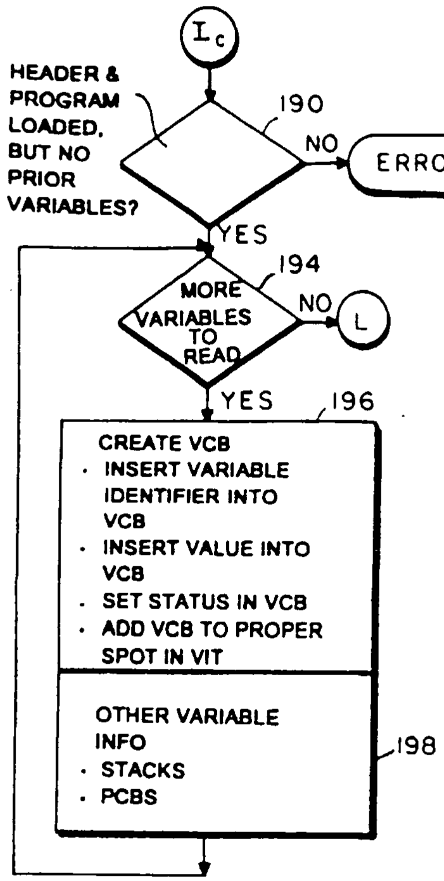


Fig. 10

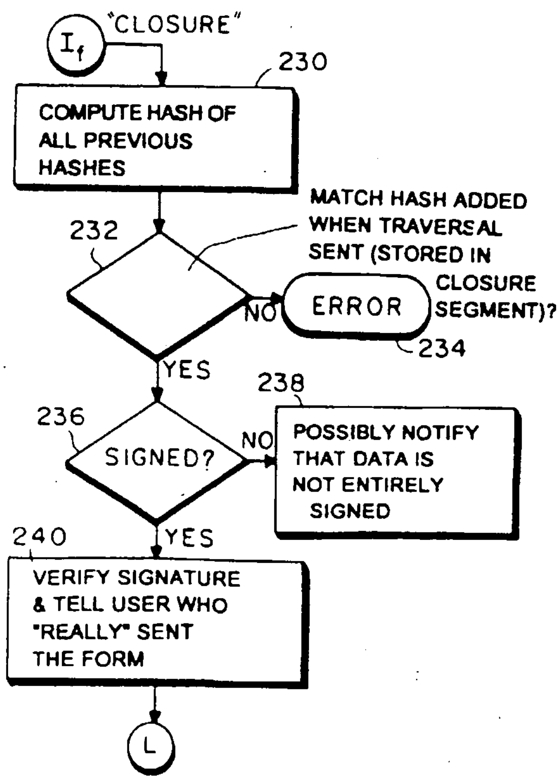


Fig. 13

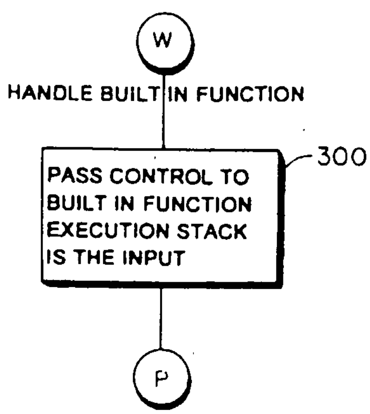
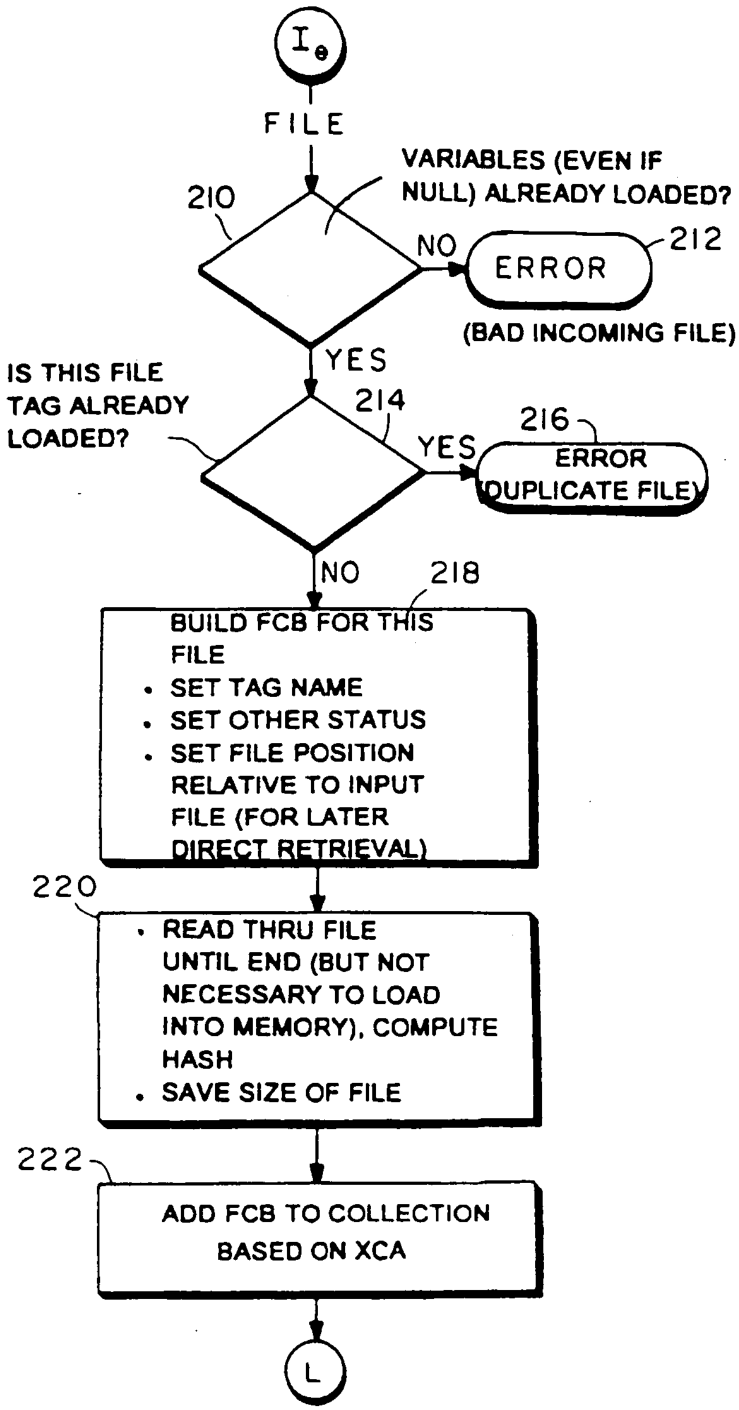


Fig. 17



I_e
FILE
VARIABLES (EVEN IF NULL) ALREADY LOADED?
210
NO
ERROR
212
(BAD INCOMING FILE)
YES
IS THIS FILE TAG ALREADY LOADED?
214
YES
216
ERROR (DUPLICATE FILE)
NO
218
BUILD FCB FOR THIS FILE
• SET TAG NAME
• SET OTHER STATUS
• SET FILE POSITION RELATIVE TO INPUT FILE (FOR LATER DIRECT RETRIEVAL)
220
• READ THRU FILE UNTIL END (BUT NOT NECESSARY TO LOAD INTO MEMORY), COMPUTE HASH
• SAVE SIZE OF FILE
222
ADD FCB TO COLLECTION BASED ON XCA
L

Fig. 12

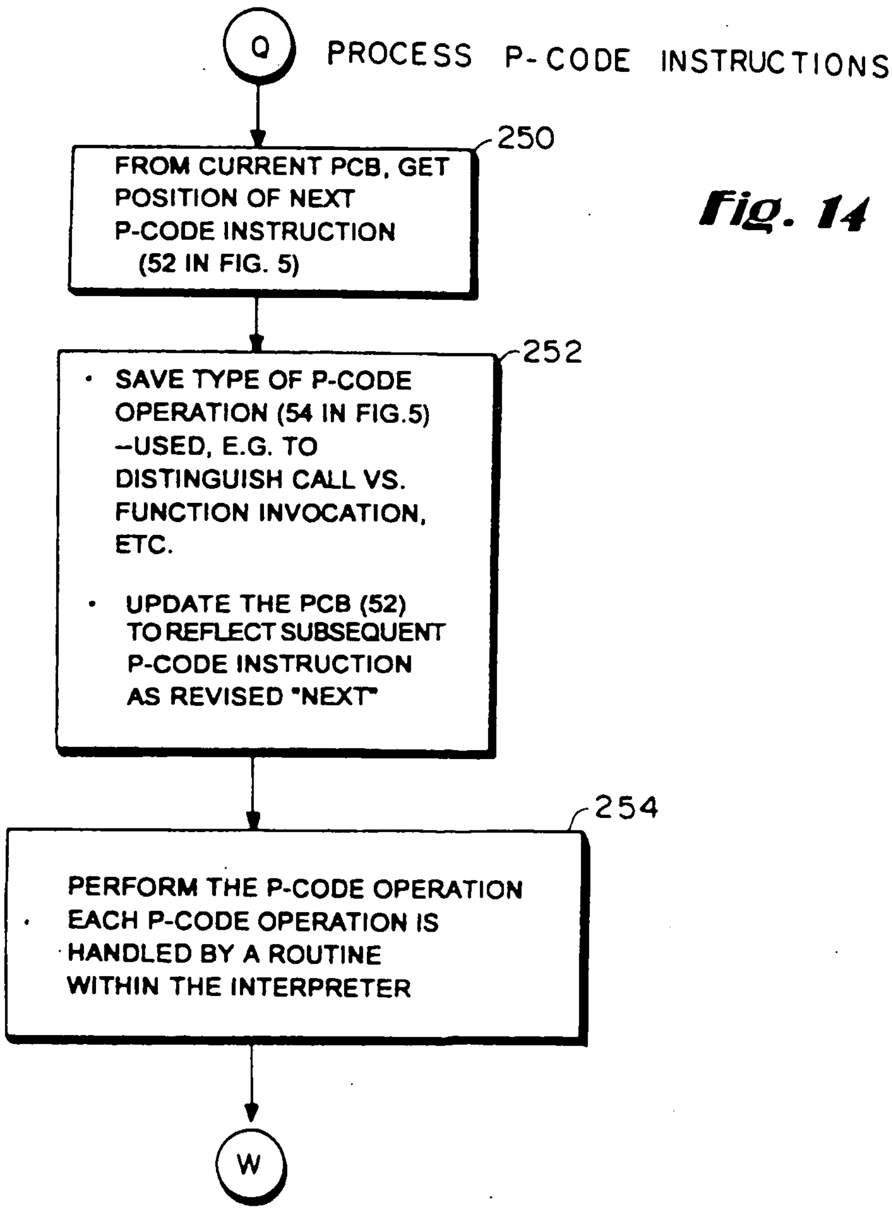
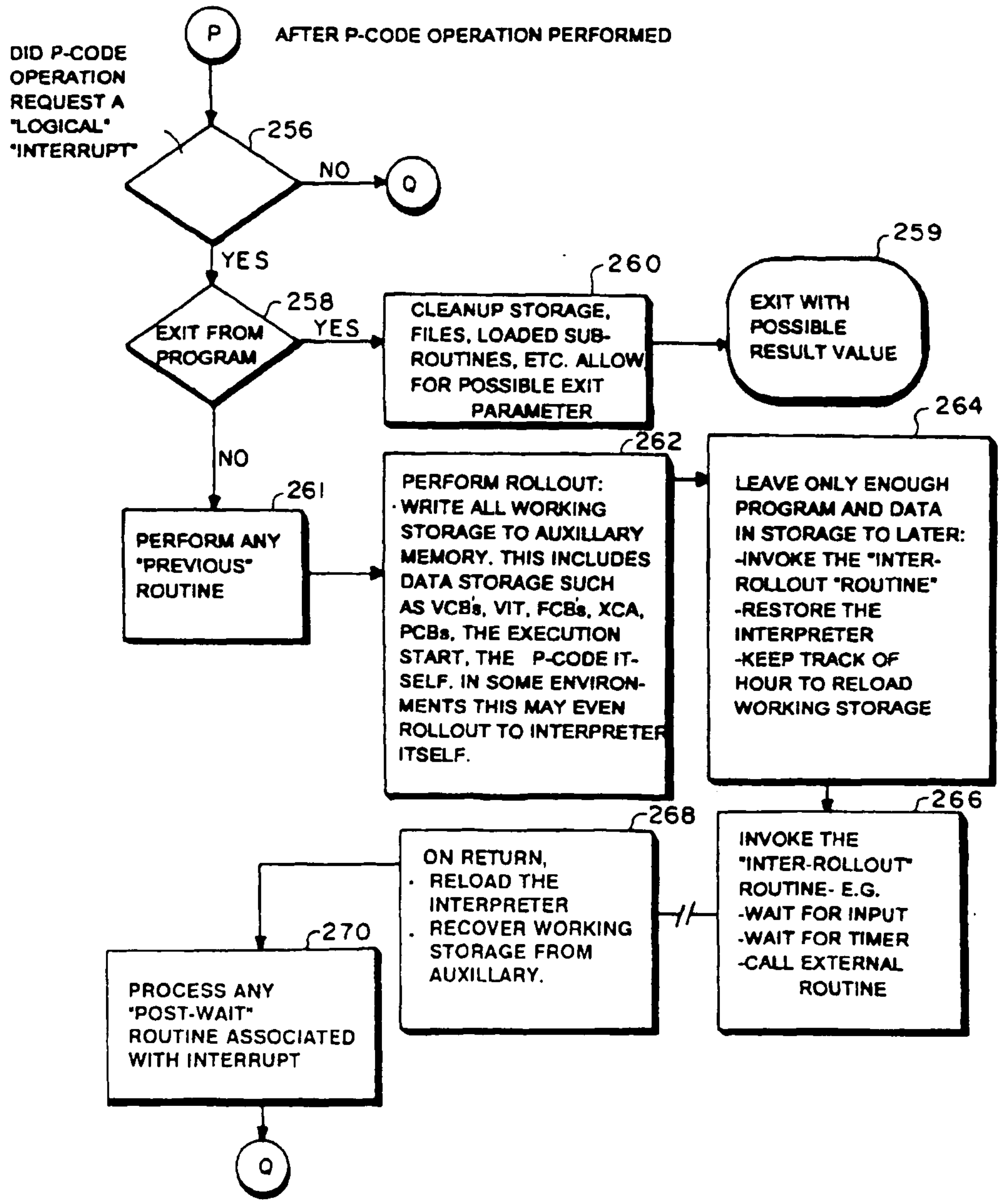

Q
PROCESS P-CODE INSTRUCTIONS
250
FROM CURRENT PCB, GET POSITION OF NEXT P-CODE INSTRUCTION (52 IN FIG. 5)
252
• SAVE TYPE OF P-CODE OPERATION (54 IN FIG.5) --USED, E.G. TO DISTINGUISH CALL VS. FUNCTION INVOCATION, ETC.
• UPDATE THE PCB (52) TO REFLECT SUBSEQUENT P-CODE INSTRUCTION AS REVISED "NEXT"
254
PERFORM THE P-CODE OPERATION
• EACH P-CODE OPERATION IS HANDLED BY A ROUTINE WITHIN THE INTERPRETER
W

Fig. 14

Fig. 15



P
AFTER P-CODE OPERATION PERFORMED
DID P-CODE OPERATION REQUEST A "LOGICAL" "INTERRUPT"
256
NO
Q
YES
258
EXIT FROM PROGRAM
YES
260
CLEANUP STORAGE, FILES, LOADED SUB-ROUTINES, ETC. ALLOW FOR POSSIBLE EXIT PARAMETER
259
EXIT WITH POSSIBLE RESULT VALUE
NO
261
PERFORM ANY "PREVIOUS" ROUTINE
262
PERFORM ROLLOUT:
· WRITE ALL WORKING STORAGE TO AUXILLARY MEMORY. THIS INCLUDES DATA STORAGE SUCH AS VCB's, VIT, FCB's, XCA, PCBs, THE EXECUTION START, THE P-CODE IT-SELF. IN SOME ENVIRON-MENTS THIS MAY EVEN ROLLOUT TO INTERPRETER ITSELF.
264
LEAVE ONLY ENOUGH PROGRAM AND DATA IN STORAGE TO LATER:
-INVOKE THE "INTER-ROLLOUT "ROUTINE"
-RESTORE THE INTERPRETER
-KEEP TRACK OF HOUR TO RELOAD WORKING STORAGE
266
INVOKE THE "INTER-ROLLOUT" ROUTINE- E.G.
-WAIT FOR INPUT
-WAIT FOR TIMER
-CALL EXTERNAL ROUTINE
268
ON RETURN,
· RELOAD THE INTERPRETER
· RECOVER WORKING STORAGE FROM AUXILLARY.
270
PROCESS ANY "POST-WAIT" ROUTINE ASSOCIATED WITH INTERRUPT
Q

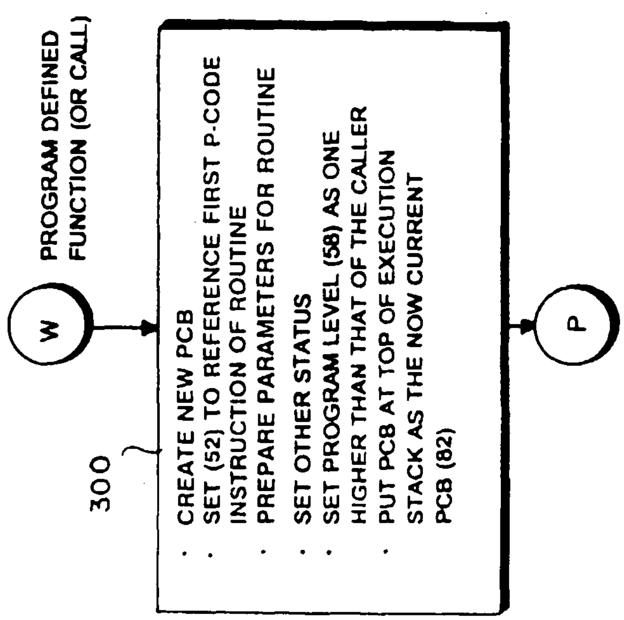


Fig. 16A

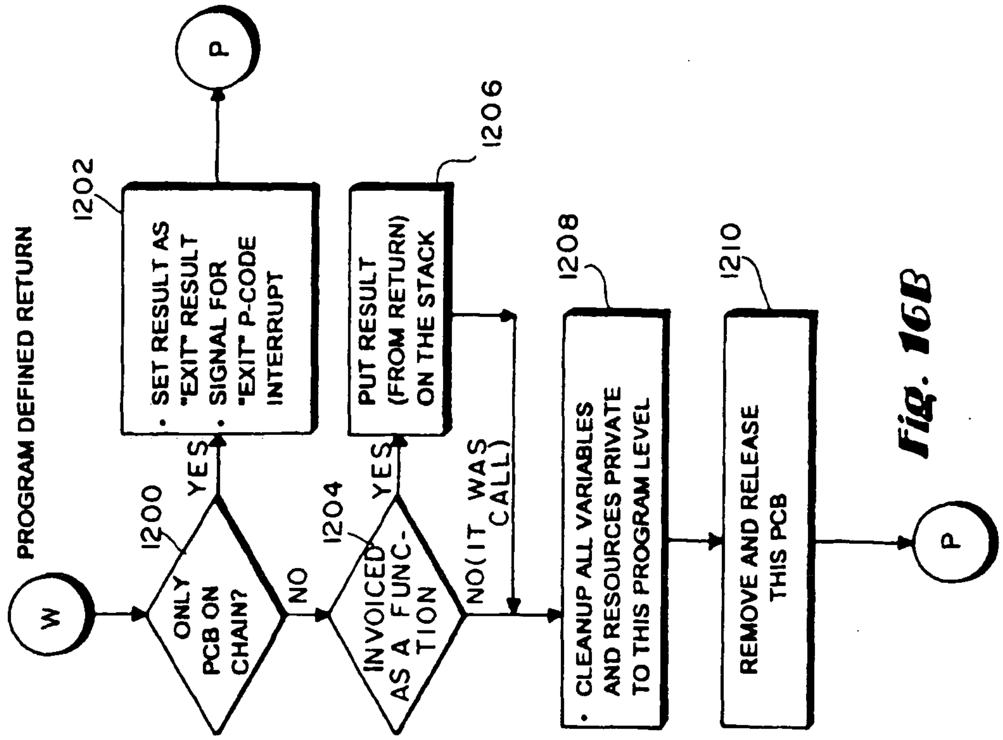


Fig. 16B

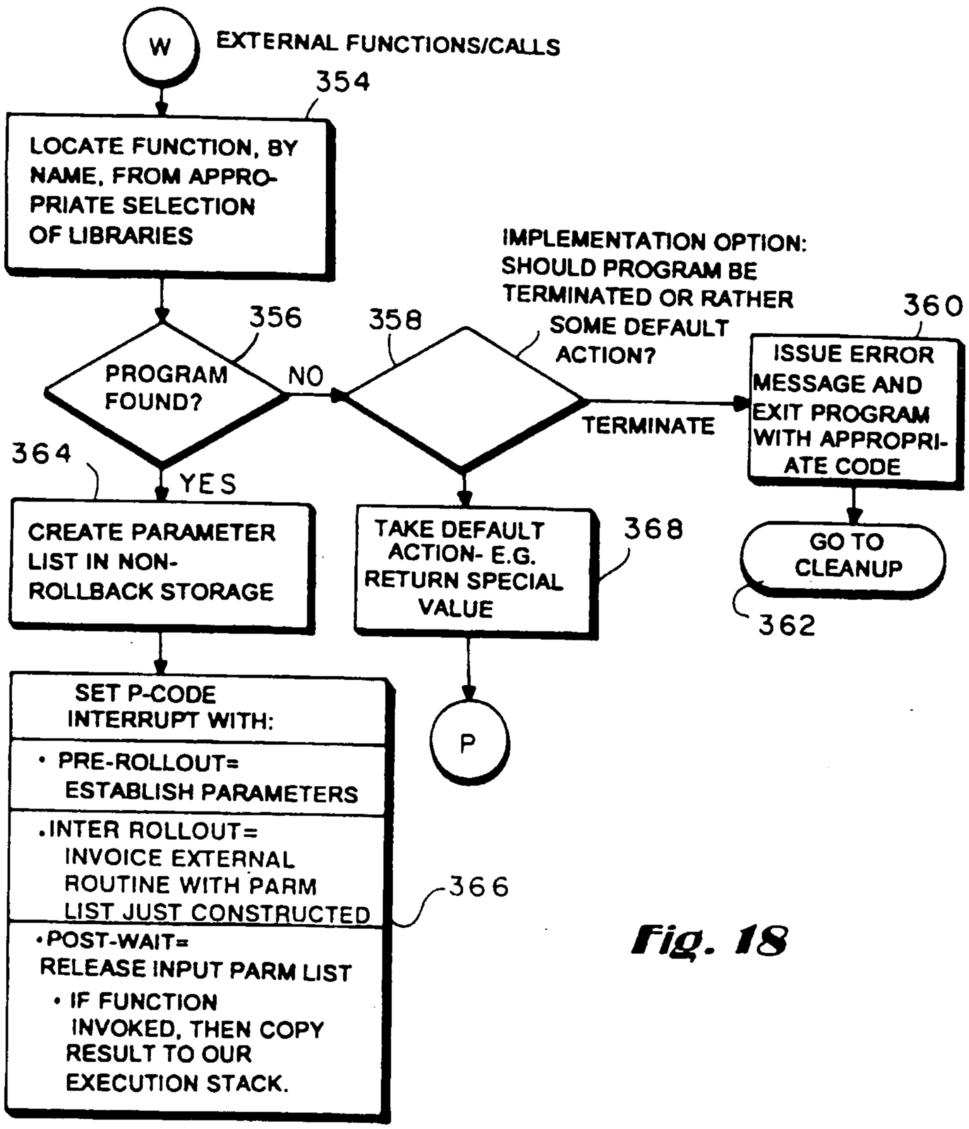

W
EXTERNAL FUNCTIONS/CALLS
354
LOCATE FUNCTION, BY NAME, FROM APPROPRIATE SELECTION OF LIBRARIES
IMPLEMENTATION OPTION: SHOULD PROGRAM BE TERMINATED OR RATHER SOME DEFAULT ACTION?
356
358
360
PROGRAM FOUND?
NO
TERMINATE
ISSUE ERROR MESSAGE AND EXIT PROGRAM WITH APPROPRIATE CODE
364
YES
CREATE PARAMETER LIST IN NON-ROLLBACK STORAGE
TAKE DEFAULT ACTION- E.G. RETURN SPECIAL VALUE
368
GO TO CLEANUP
362
SET P-CODE INTERRUPT WITH:
• PRE-ROLLOUT= ESTABLISH PARAMETERS
.INTER ROLLOUT= INVOICE EXTERNAL ROUTINE WITH PARM LIST JUST CONSTRUCTED
•POST-WAIT= RELEASE INPUT PARM LIST
• IF FUNCTION INVOKED, THEN COPY RESULT TO OUR EXECUTION STACK.
P
366

Fig. 18

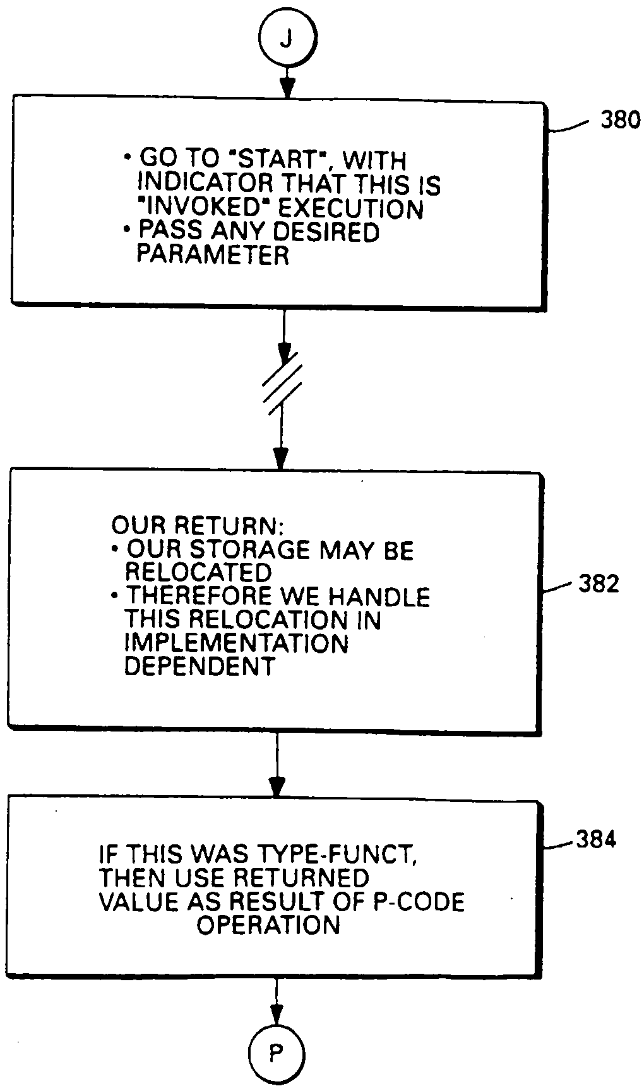

J
• GO TO "START", WITH INDICATOR THAT THIS IS "INVOKED" EXECUTION
• PASS ANY DESIRED PARAMETER
380
OUR RETURN:
• OUR STORAGE MAY BE RELOCATED
• THEREFORE WE HANDLE THIS RELOCATION IN IMPLEMENTATION DEPENDENT
382
IF THIS WAS TYPE-FUNCT, THEN USE RETURNED VALUE AS RESULT OF P-CODE OPERATION
384
P

Fig. 19

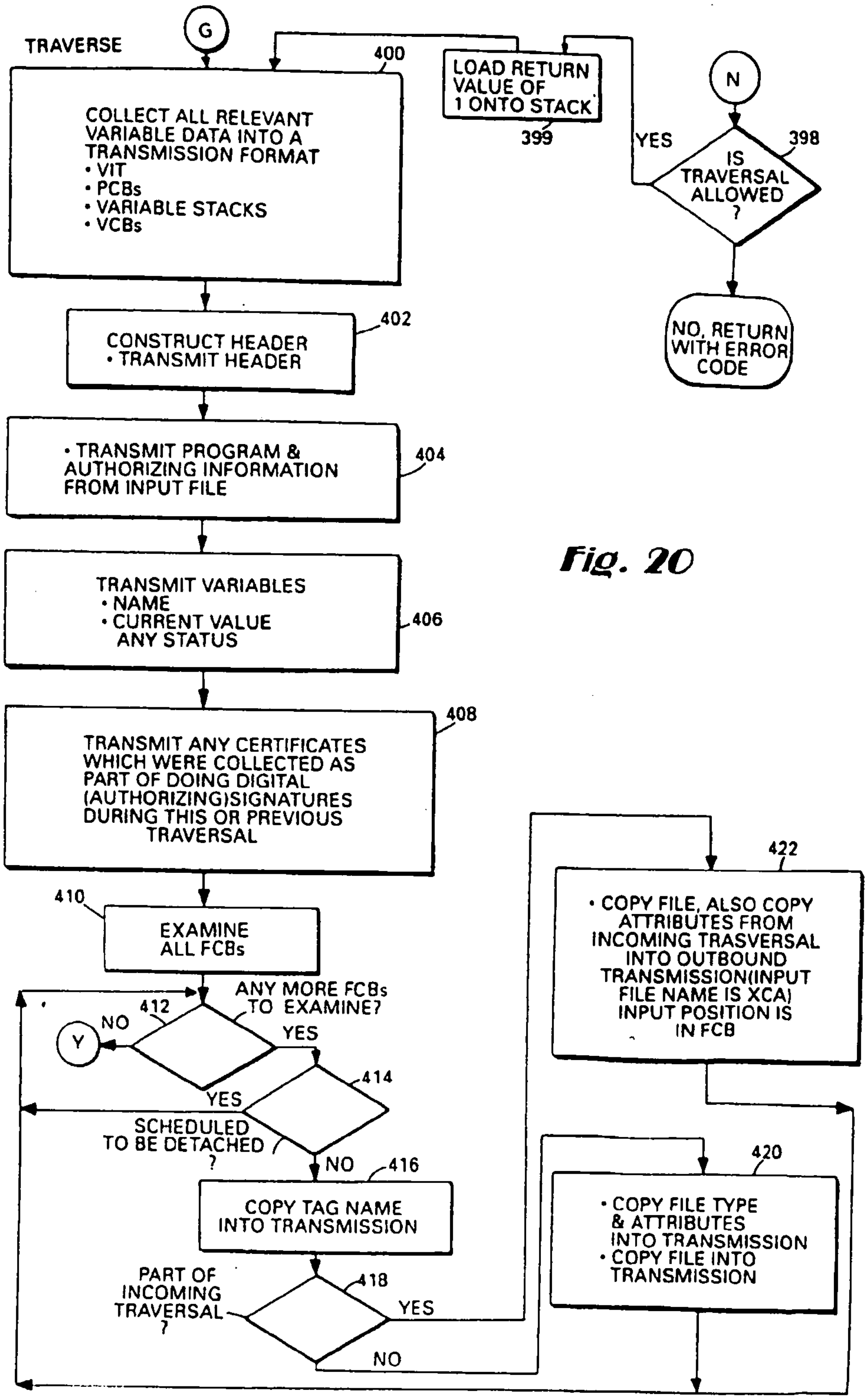

TRAVERSE
G
400
COLLECT ALL RELEVANT VARIABLE DATA INTO A TRANSMISSION FORMAT
• VIT
• PCBs
• VARIABLE STACKS
• VCBs
LOAD RETURN VALUE OF 1 ONTO STACK
399
N
YES
398
IS TRAVERSAL ALLOWED ?
NO, RETURN WITH ERROR CODE
402
CONSTRUCT HEADER
• TRANSMIT HEADER
404
• TRANSMIT PROGRAM & AUTHORIZING INFORMATION FROM INPUT FILE
TRANSMIT VARIABLES
• NAME
• CURRENT VALUE ANY STATUS
406
408
TRANSMIT ANY CERTIFICATES WHICH WERE COLLECTED AS PART OF DOING DIGITAL (AUTHORIZING)SIGNATURES DURING THIS OR PREVIOUS TRAVERSAL
410
EXAMINE ALL FCBs
412
ANY MORE FCBs TO EXAMINE?
NO
Y
YES
414
YES
SCHEDULED TO BE DETACHED ?
NO
416
COPY TAG NAME INTO TRANSMISSION
PART OF INCOMING TRAVERSAL ?
418
YES
NO
422
• COPY FILE, ALSO COPY ATTRIBUTES FROM INCOMING TRASVERSAL INTO OUTBOUND TRANSMISSION(INPUT FILE NAME IS XCA) INPUT POSITION IS IN FCB
420
• COPY FILE TYPE & ATTRIBUTES INTO TRANSMISSION
• COPY FILE INTO TRANSMISSION

Fig. 20

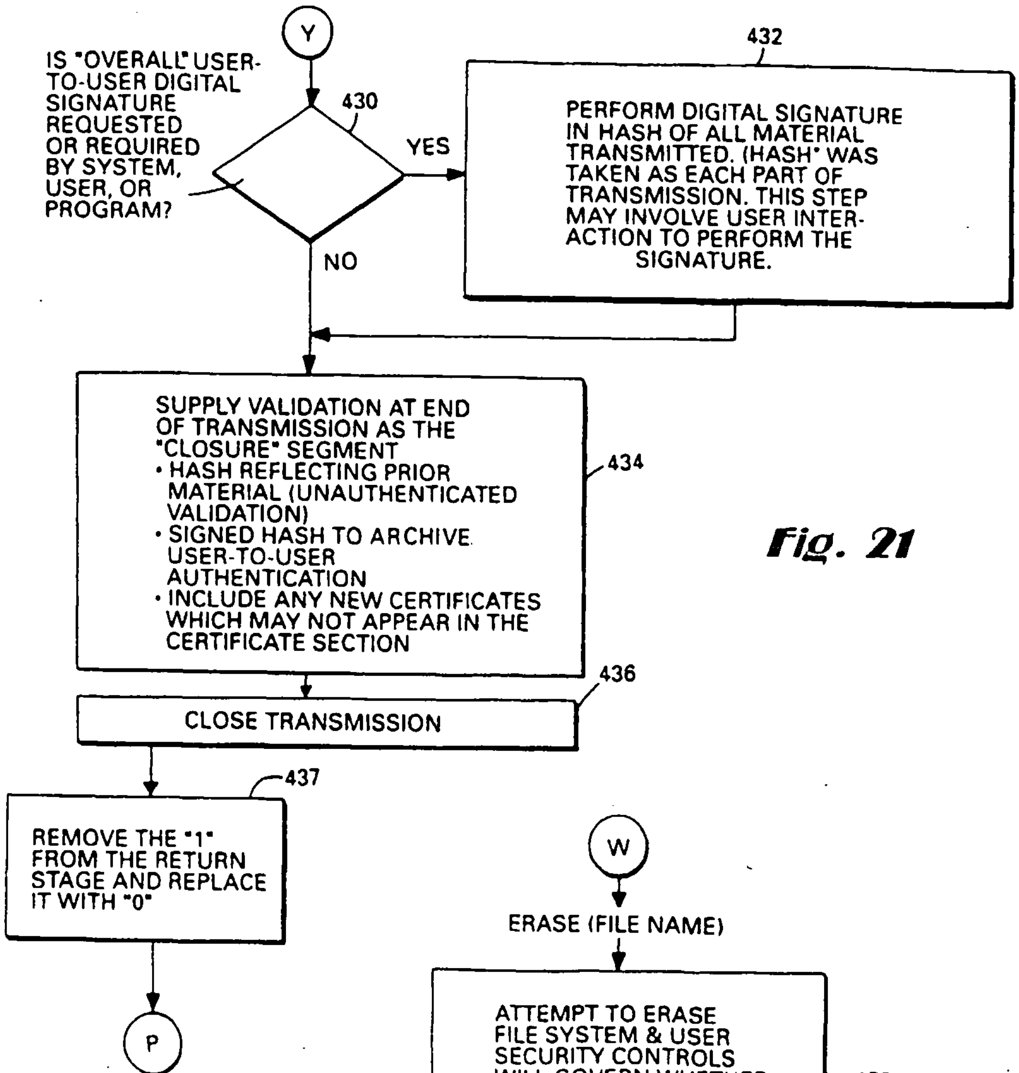

Y
IS "OVERALL" USER-TO-USER DIGITAL SIGNATURE REQUESTED OR REQUIRED BY SYSTEM, USER, OR PROGRAM?
430
YES
NO
432
PERFORM DIGITAL SIGNATURE IN HASH OF ALL MATERIAL TRANSMITTED. (HASH" WAS TAKEN AS EACH PART OF TRANSMISSION. THIS STEP MAY INVOLVE USER INTER-ACTION TO PERFORM THE SIGNATURE.
SUPPLY VALIDATION AT END OF TRANSMISSION AS THE "CLOSURE" SEGMENT
• HASH REFLECTING PRIOR MATERIAL (UNAUTHENTICATED VALIDATION)
• SIGNED HASH TO ARCHIVE USER-TO-USER AUTHENTICATION
• INCLUDE ANY NEW CERTIFICATES WHICH MAY NOT APPEAR IN THE CERTIFICATE SECTION
434
CLOSE TRANSMISSION
436
REMOVE THE "1" FROM THE RETURN STAGE AND REPLACE IT WITH "0"
437
P

Fig. 21

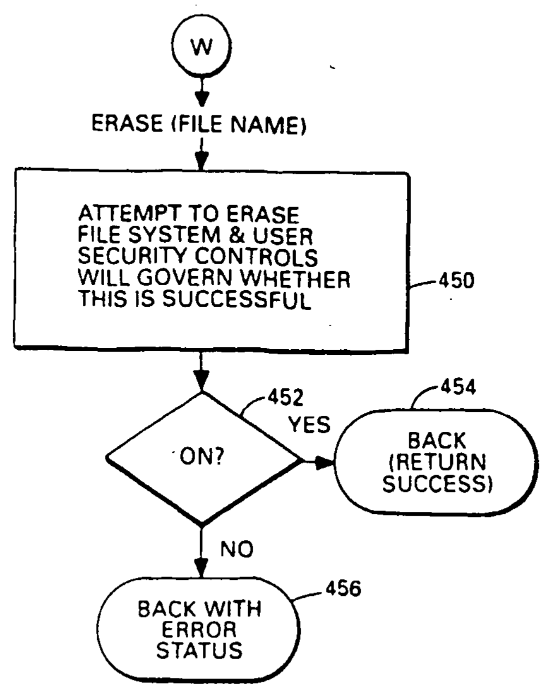

W
ERASE (FILE NAME)
ATTEMPT TO ERASE FILE SYSTEM & USER SECURITY CONTROLS WILL GOVERN WHETHER THIS IS SUCCESSFUL
450
452
ON?
YES
NO
454
BACK (RETURN SUCCESS)
456
BACK WITH ERROR STATUS

Fig. 23

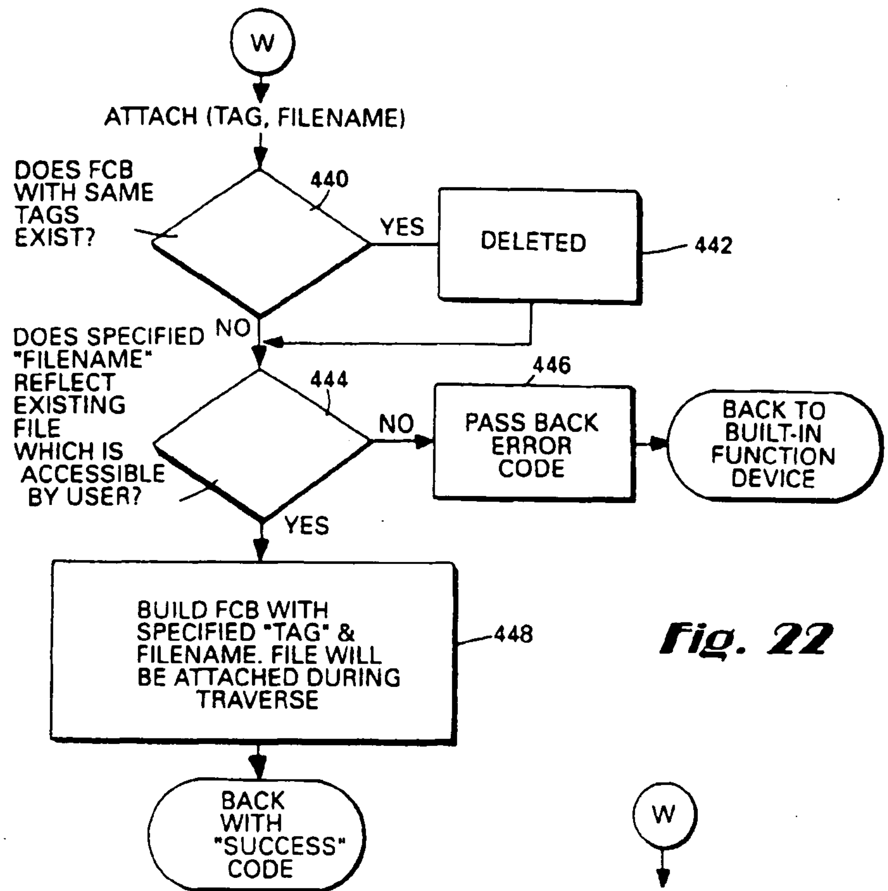

W
ATTACH (TAG, FILENAME)
DOES FCB WITH SAME TAGS EXIST?
440
YES
DELETED
442
NO
DOES SPECIFIED "FILENAME" REFLECT EXISTING FILE WHICH IS ACCESSIBLE BY USER?
444
NO
446
PASS BACK ERROR CODE
BACK TO BUILT-IN FUNCTION DEVICE
YES
BUILD FCB WITH SPECIFIED "TAG" & FILENAME. FILE WILL BE ATTACHED DURING TRAVERSE
448
BACK WITH "SUCCESS" CODE

Fig. 22

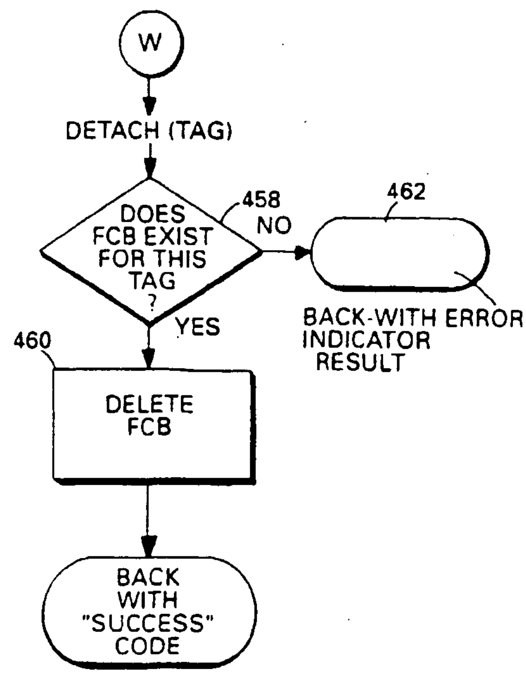

W
DETACH (TAG)
DOES FCB EXIST FOR THIS TAG ?
458
NO
462
BACK-WITH ERROR INDICATOR RESULT
YES
460
DELETE FCB
BACK WITH "SUCCESS" CODE

Fig. 24

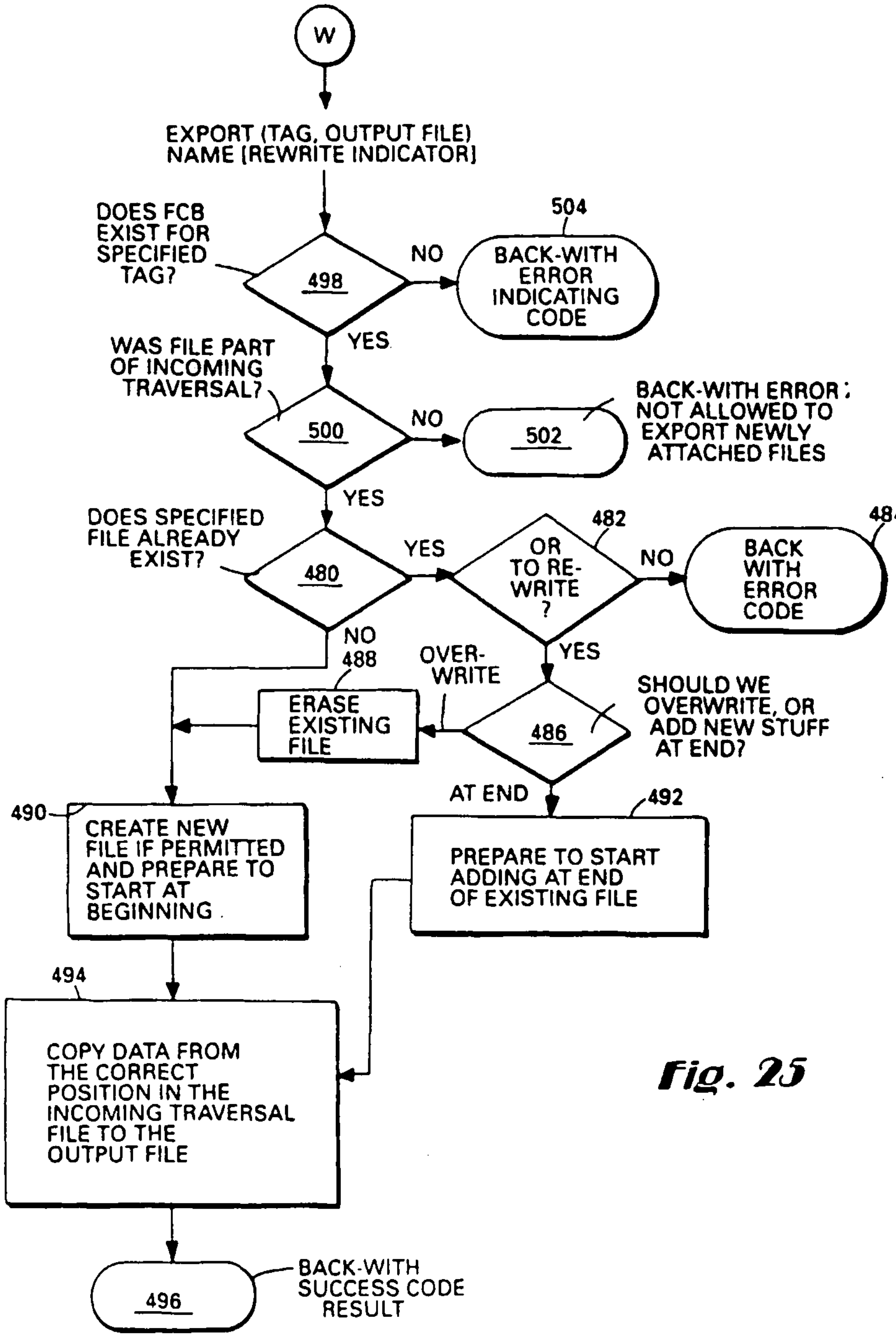

W
EXPORT (TAG, OUTPUT FILE) NAME [REWRITE INDICATOR]
DOES FCB EXIST FOR SPECIFIED TAG?
498
NO
504
BACK-WITH ERROR INDICATING CODE
YES
WAS FILE PART OF INCOMING TRAVERSAL?
500
NO
502
BACK-WITH ERROR: NOT ALLOWED TO EXPORT NEWLY ATTACHED FILES
YES
DOES SPECIFIED FILE ALREADY EXIST?
480
YES
482
OR TO RE-WRITE ?
NO
484
BACK WITH ERROR CODE
NO
488
OVER-WRITE
YES
ERASE EXISTING FILE
486
SHOULD WE OVERWRITE, OR ADD NEW STUFF AT END?
AT END
490
CREATE NEW FILE IF PERMITTED AND PREPARE TO START AT BEGINNING
492
PREPARE TO START ADDING AT END OF EXISTING FILE
494
COPY DATA FROM THE CORRECT POSITION IN THE INCOMING TRAVERSAL FILE TO THE OUTPUT FILE
496
BACK-WITH SUCCESS CODE RESULT

Fig. 25

Fig. 26

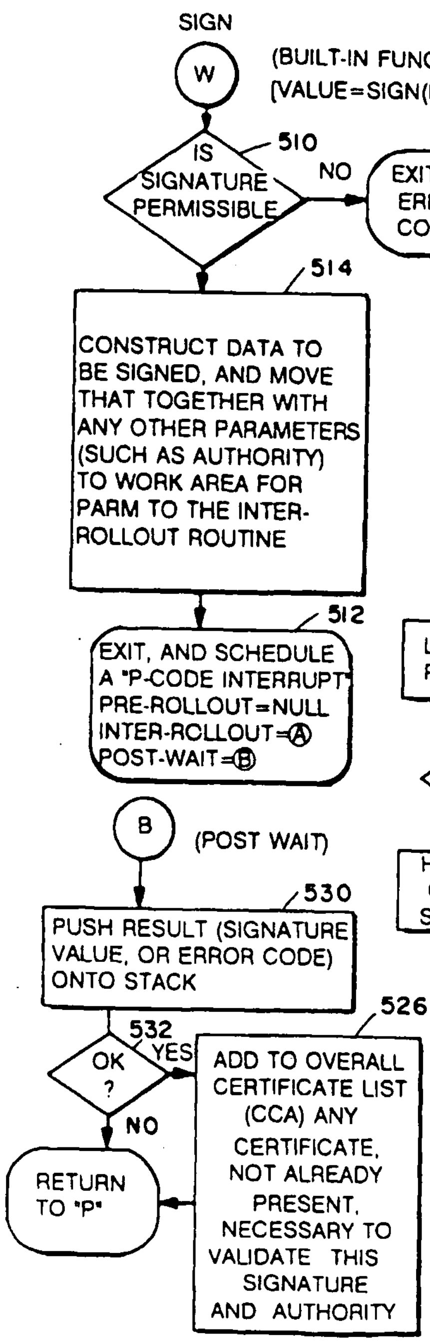
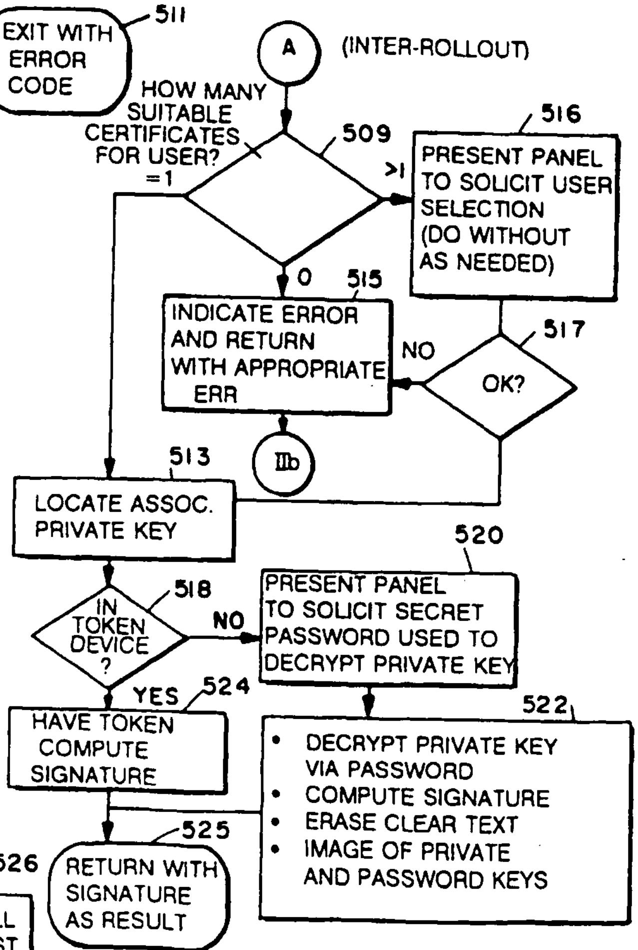

SIGN
W
(BUILT-IN FUNCTION)
[VALUE=SIGN(DATA, PARMS...)]
510
IS SIGNATURE PERMISSIBLE
NO
511
EXIT WITH ERROR CODE
514
CONSTRUCT DATA TO BE SIGNED, AND MOVE THAT TOGETHER WITH ANY OTHER PARAMETERS (SUCH AS AUTHORITY) TO WORK AREA FOR PARM TO THE INTER-ROLLOUT ROUTINE
512
EXIT, AND SCHEDULE A "P-CODE INTERRUPT" PRE-ROLLOUT=NULL INTER-ROLLOUT=Ⓐ POST-WAIT=Ⓑ
B
(POST WAIT)
530
PUSH RESULT (SIGNATURE VALUE, OR ERROR CODE) ONTO STACK
532
OK ?
YES
NO
526
ADD TO OVERALL CERTIFICATE LIST (CCA) ANY CERTIFICATE, NOT ALREADY PRESENT, NECESSARY TO VALIDATE THIS SIGNATURE AND AUTHORITY
RETURN TO "P"

Fig. 27


A
(INTER-ROLLOUT)
HOW MANY SUITABLE CERTIFICATES FOR USER?
509
=1
>1
0
516
PRESENT PANEL TO SOLICIT USER SELECTION (DO WITHOUT AS NEEDED)
515
INDICATE ERROR AND RETURN WITH APPROPRIATE ERR
517
OK?
NO
IIb
513
LOCATE ASSOC. PRIVATE KEY
518
IN TOKEN DEVICE ?
NO
YES
520
PRESENT PANEL TO SOLICIT SECRET PASSWORD USED TO DECRYPT PRIVATE KEY
524
HAVE TOKEN COMPUTE SIGNATURE
522
• DECRYPT PRIVATE KEY VIA PASSWORD
• COMPUTE SIGNATURE
• ERASE CLEAR TEXT
• IMAGE OF PRIVATE AND PASSWORD KEYS
525
RETURN WITH SIGNATURE AS RESULT

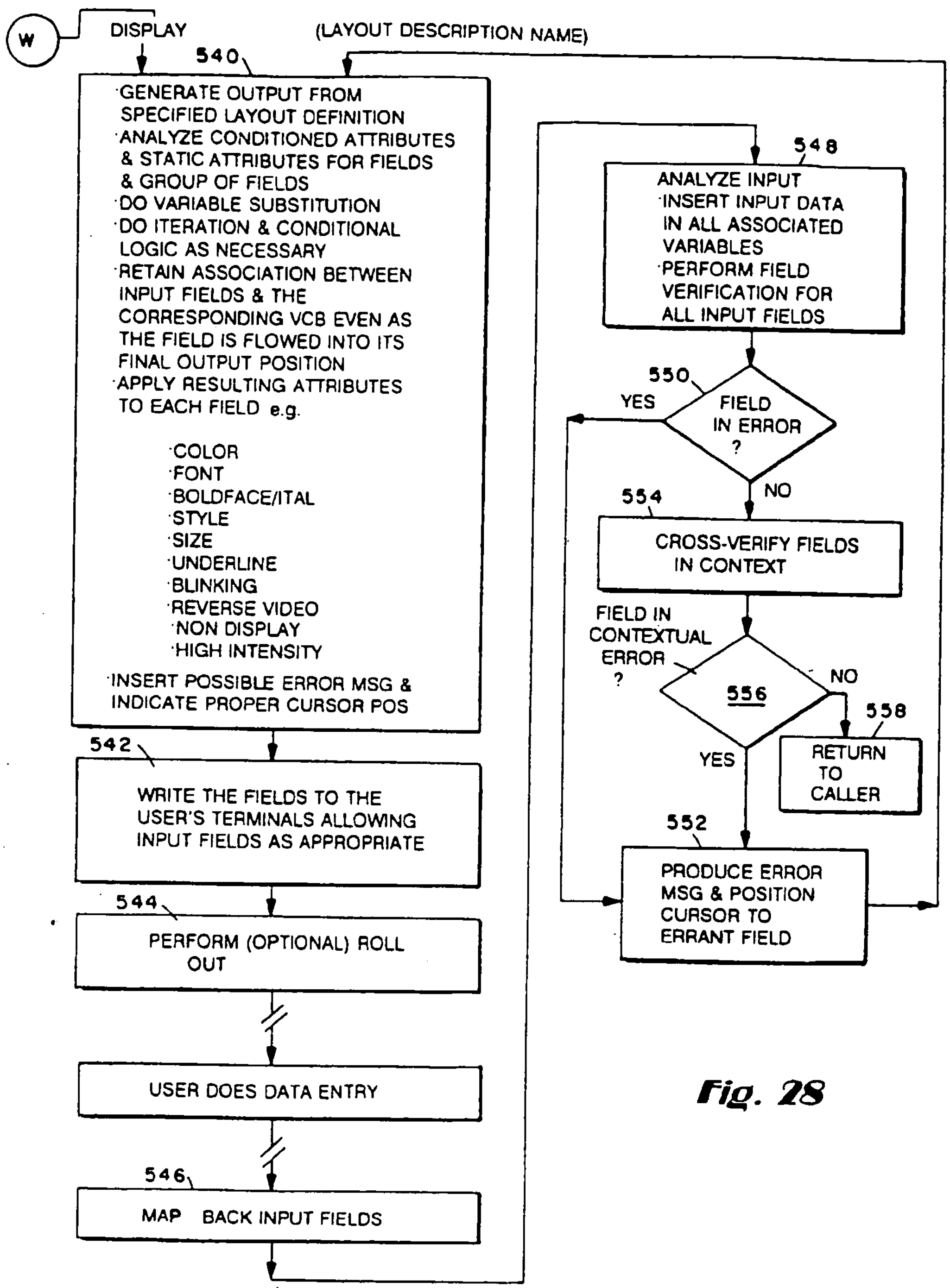

W
DISPLAY
(LAYOUT DESCRIPTION NAME)
540
·GENERATE OUTPUT FROM SPECIFIED LAYOUT DEFINITION
·ANALYZE CONDITIONED ATTRIBUTES & STATIC ATTRIBUTES FOR FIELDS & GROUP OF FIELDS
·DO VARIABLE SUBSTITUTION
·DO ITERATION & CONDITIONAL LOGIC AS NECESSARY
·RETAIN ASSOCIATION BETWEEN INPUT FIELDS & THE CORRESPONDING VCB EVEN AS THE FIELD IS FLOWED INTO ITS FINAL OUTPUT POSITION
·APPLY RESULTING ATTRIBUTES TO EACH FIELD e.g.
·COLOR
·FONT
·BOLDFACE/ITAL
·STYLE
·SIZE
·UNDERLINE
·BLINKING
·REVERSE VIDEO
·NON DISPLAY
·HIGH INTENSITY
·INSERT POSSIBLE ERROR MSG & INDICATE PROPER CURSOR POS
542
WRITE THE FIELDS TO THE USER'S TERMINALS ALLOWING INPUT FIELDS AS APPROPRIATE
544
PERFORM (OPTIONAL) ROLL OUT
USER DOES DATA ENTRY
546
MAP BACK INPUT FIELDS
548
ANALYZE INPUT
·INSERT INPUT DATA IN ALL ASSOCIATED VARIABLES
·PERFORM FIELD VERIFICATION FOR ALL INPUT FIELDS
550
FIELD IN ERROR ?
YES
NO
554
CROSS-VERIFY FIELDS IN CONTEXT
FIELD IN CONTEXTUAL ERROR ?
556
NO
YES
558
RETURN TO CALLER
552
PRODUCE ERROR MSG & POSITION CURSOR TO ERRANT FIELD

Fig. 28

Fig. 29

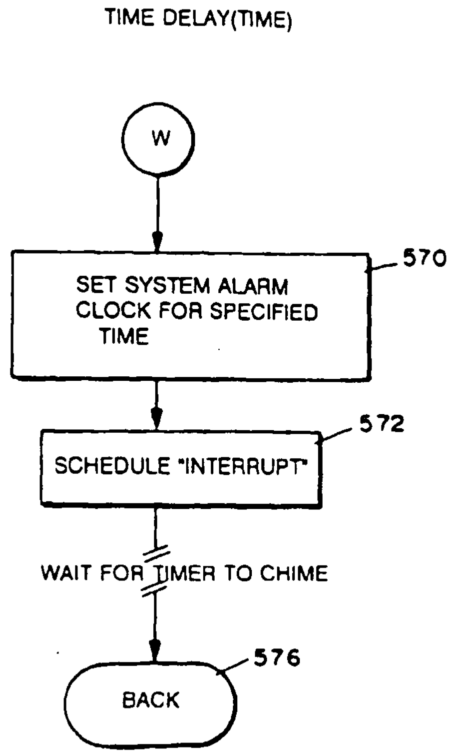

TIME DELAY(TIME)
W
SET SYSTEM ALARM CLOCK FOR SPECIFIED TIME
570
SCHEDULE "INTERRUPT"
572
WAIT FOR TIMER TO CHIME
576
BACK

Fig. 30

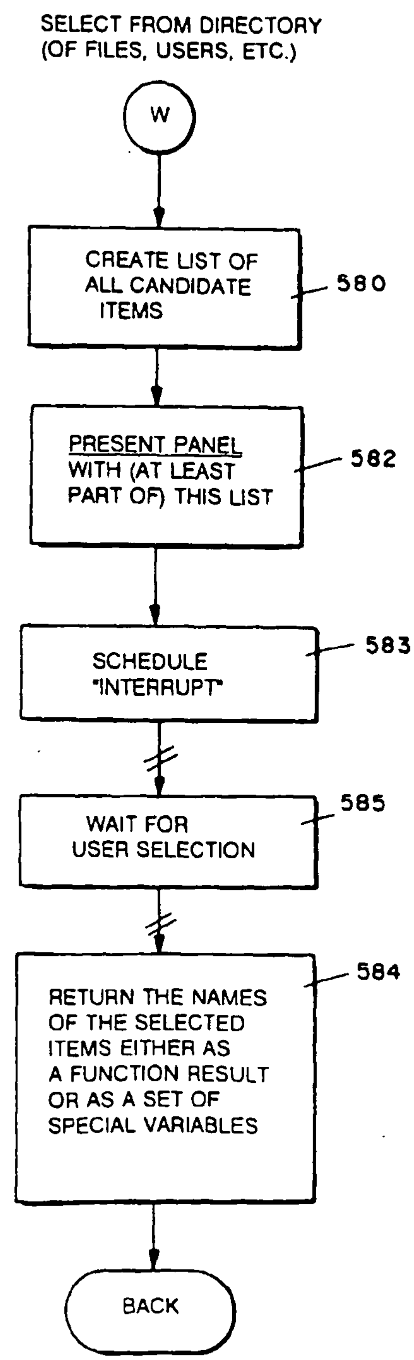

SELECT FROM DIRECTORY (OF FILES, USERS, ETC.)
W
CREATE LIST OF ALL CANDIDATE ITEMS
580
PRESENT PANEL WITH (AT LEAST PART OF) THIS LIST
582
SCHEDULE "INTERRUPT"
583
WAIT FOR USER SELECTION
585
RETURN THE NAMES OF THE SELECTED ITEMS EITHER AS A FUNCTION RESULT OR AS A SET OF SPECIAL VARIABLES
584
BACK

Fig. 31

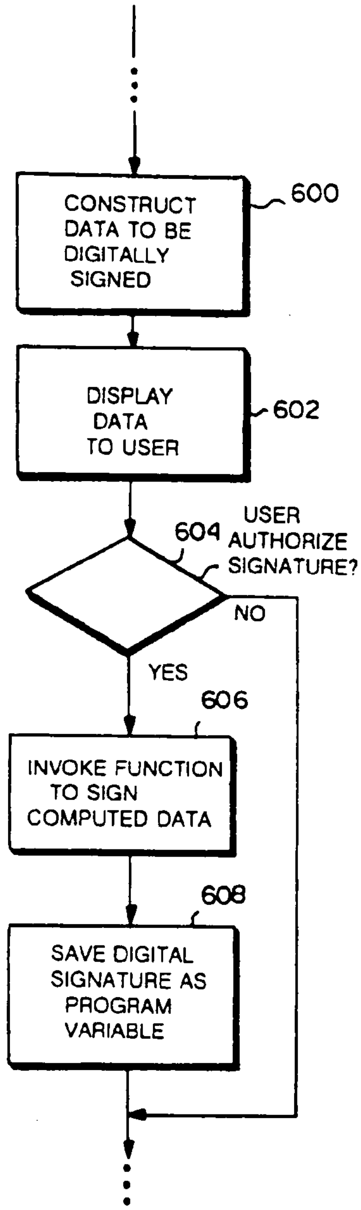
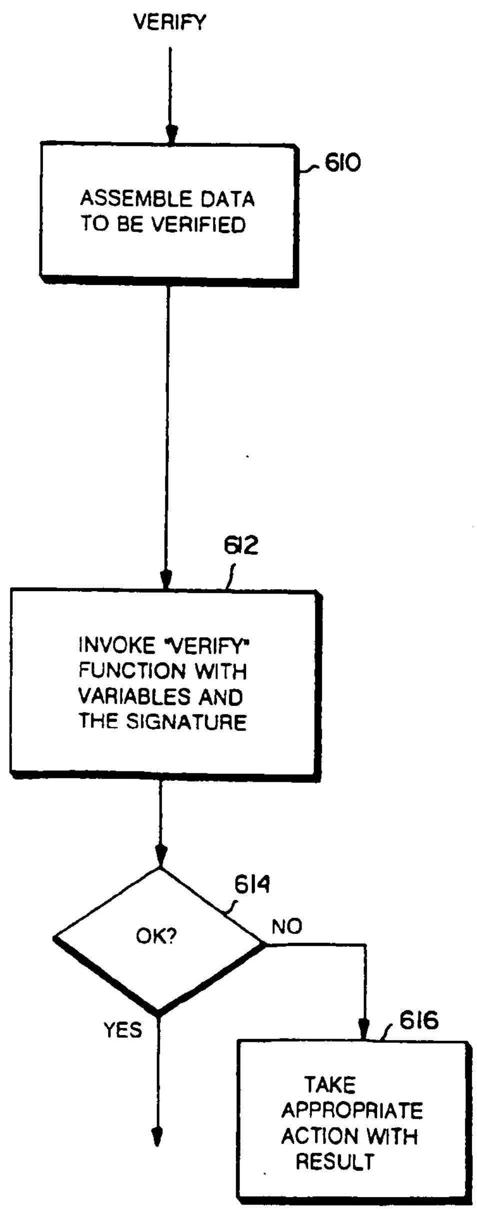

CONSTRUCT DATA TO BE DIGITALLY SIGNED
600
DISPLAY DATA TO USER
602
604
USER AUTHORIZE SIGNATURE?
NO
YES
606
INVOKE FUNCTION TO SIGN COMPUTED DATA
608
SAVE DIGITAL SIGNATURE AS PROGRAM VARIABLE

Fig. 32


VERIFY
ASSEMBLE DATA TO BE VERIFIED
610
612
INVOKE "VERIFY" FUNCTION WITH VARIABLES AND THE SIGNATURE
614
OK?
NO
YES
616
TAKE APPROPRIATE ACTION WITH RESULT

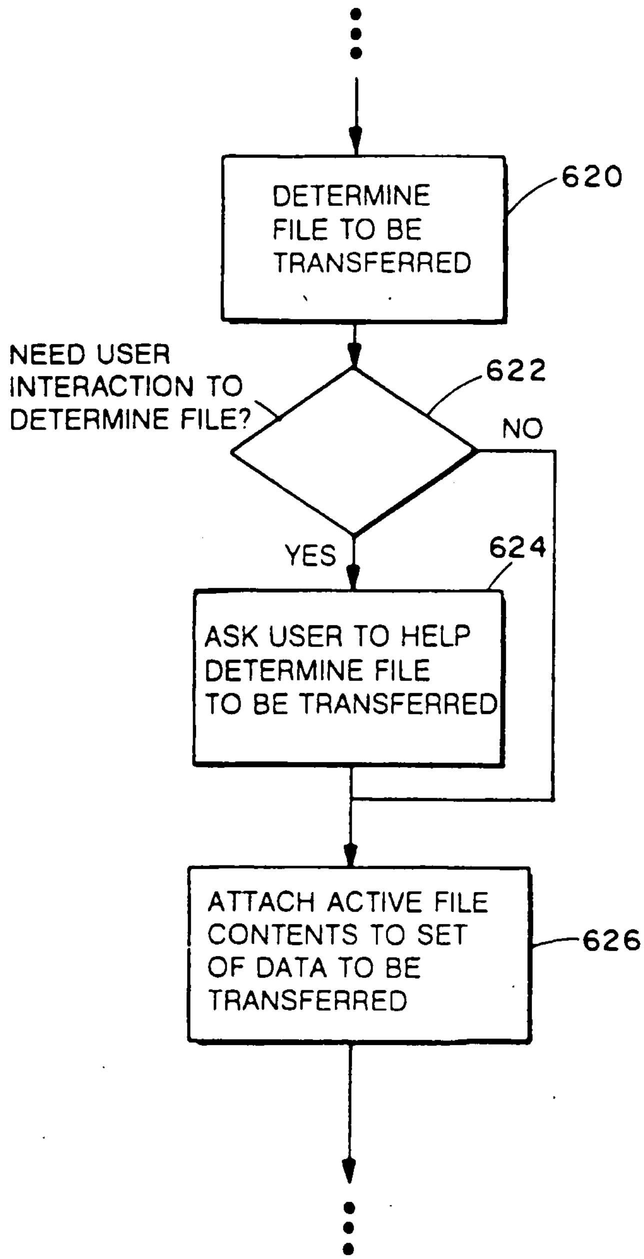

DETERMINE FILE TO BE TRANSFERRED
620
NEED USER INTERACTION TO DETERMINE FILE?
622
NO
YES
624
ASK USER TO HELP DETERMINE FILE TO BE TRANSFERRED
ATTACH ACTIVE FILE CONTENTS TO SET OF DATA TO BE TRANSFERRED
626

Fig. 33

Fig. 34

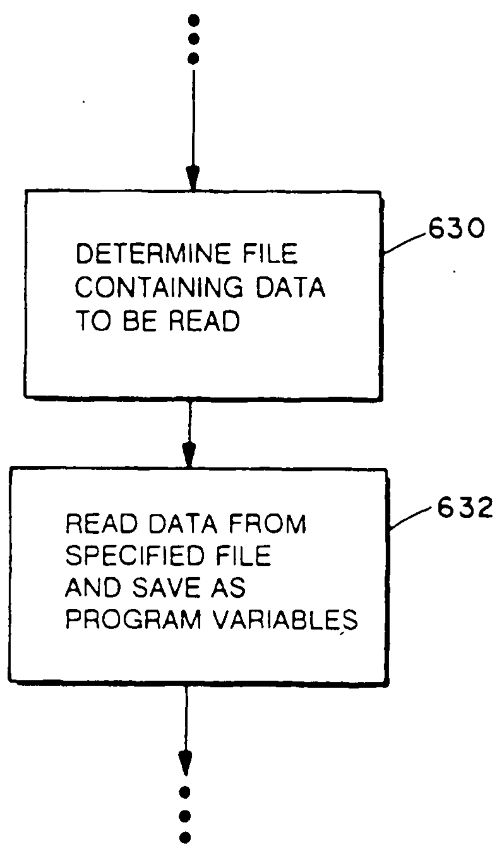
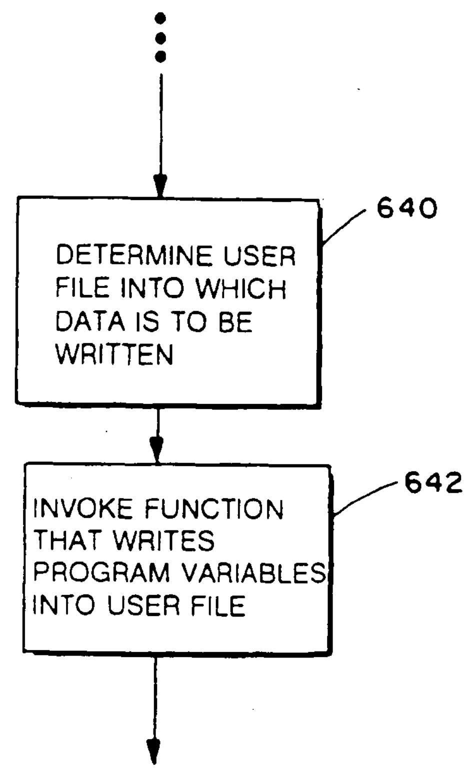

DETERMINE FILE CONTAINING DATA TO BE READ
630
READ DATA FROM SPECIFIED FILE AND SAVE AS PROGRAM VARIABLES
632

Fig. 35


DETERMINE USER FILE INTO WHICH DATA IS TO BE WRITTEN
640
INVOKE FUNCTION THAT WRITES PROGRAM VARIABLES INTO USER FILE
642

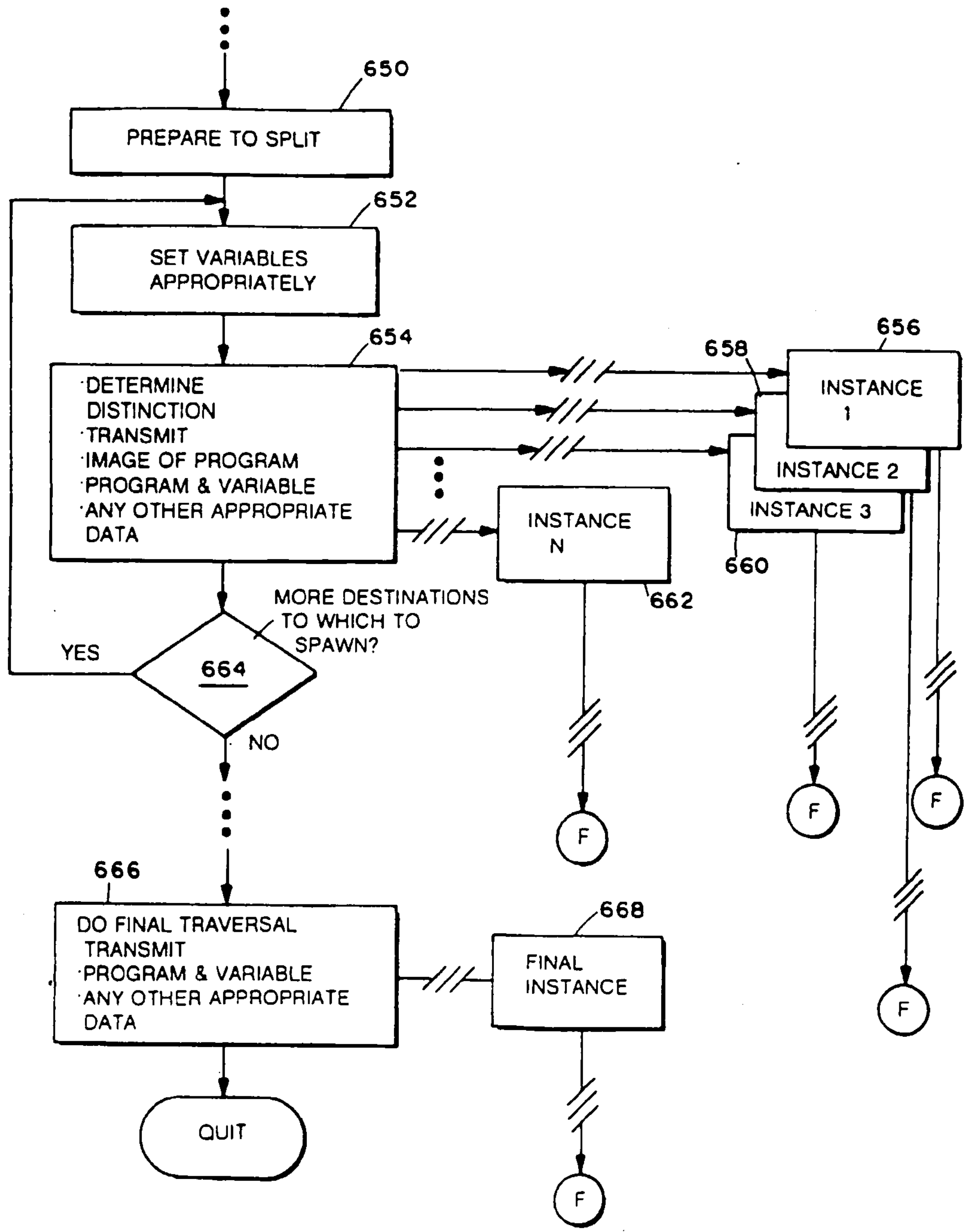

650
PREPARE TO SPLIT
652
SET VARIABLES APPROPRIATELY
654
·DETERMINE DISTINCTION
·TRANSMIT
·IMAGE OF PROGRAM
·PROGRAM & VARIABLE
·ANY OTHER APPROPRIATE DATA
658
656
INSTANCE 1
INSTANCE 2
INSTANCE 3
660
INSTANCE N
662
MORE DESTINATIONS TO WHICH TO SPAWN?
664
YES
NO
F
F
F
F
666
DO FINAL TRAVERSAL
TRANSMIT
·PROGRAM & VARIABLE
·ANY OTHER APPROPRIATE DATA
668
FINAL INSTANCE
QUIT
F

Fig. 36

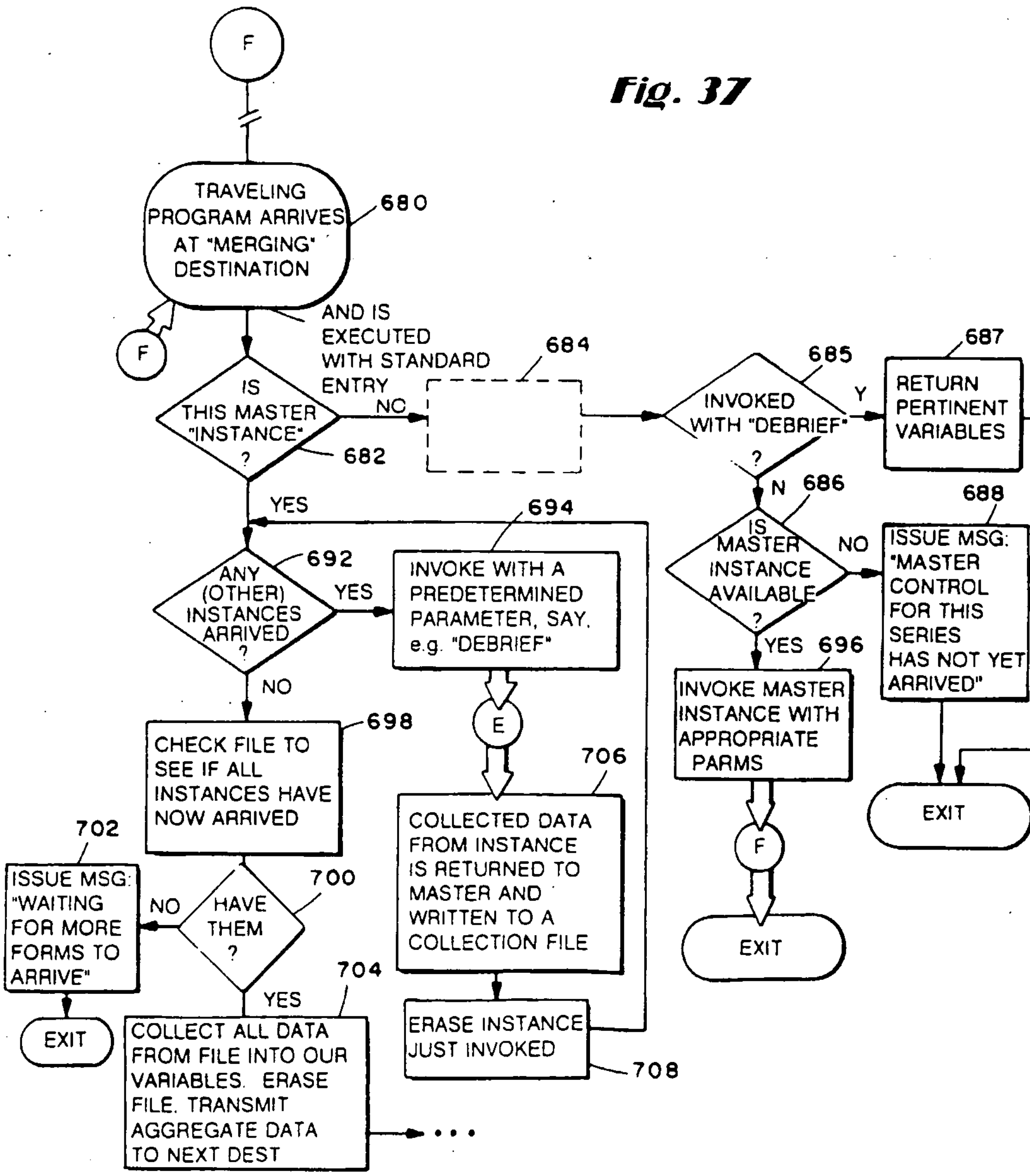

F
TRAVELING PROGRAM ARRIVES AT "MERGING" DESTINATION
680
AND IS EXECUTED WITH STANDARD ENTRY
F
IS THIS MASTER "INSTANCE" ?
682
NO
684
INVOKED WITH "DEBRIEF" ?
685
Y
RETURN PERTINENT VARIABLES
687
N
686
IS MASTER INSTANCE AVAILABLE ?
NO
ISSUE MSG: "MASTER CONTROL FOR THIS SERIES HAS NOT YET ARRIVED"
688
YES
696
INVOKE MASTER INSTANCE WITH APPROPRIATE PARMS
F
EXIT
EXIT
YES
694
ANY (OTHER) INSTANCES ARRIVED ?
692
YES
INVOKE WITH A PREDETERMINED PARAMETER, SAY, e.g. "DEBRIEF"
E
706
COLLECTED DATA FROM INSTANCE IS RETURNED TO MASTER AND WRITTEN TO A COLLECTION FILE
ERASE INSTANCE JUST INVOKED
708
NO
CHECK FILE TO SEE IF ALL INSTANCES HAVE NOW ARRIVED
698
HAVE THEM ?
700
NO
ISSUE MSG: "WAITING FOR MORE FORMS TO ARRIVE"
702
EXIT
YES
704
COLLECT ALL DATA FROM FILE INTO OUR VARIABLES. ERASE FILE. TRANSMIT AGGREGATE DATA TO NEXT DEST
• • •

Fig. 37

Fig. 38

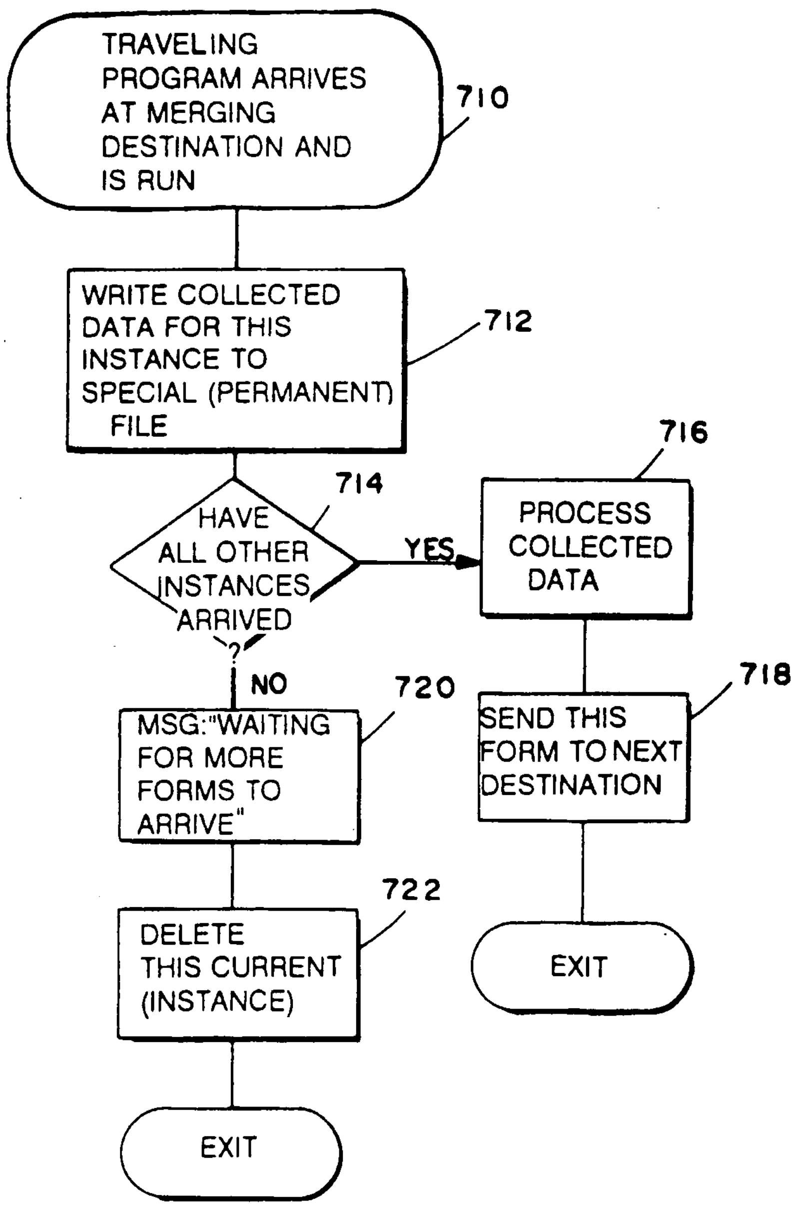
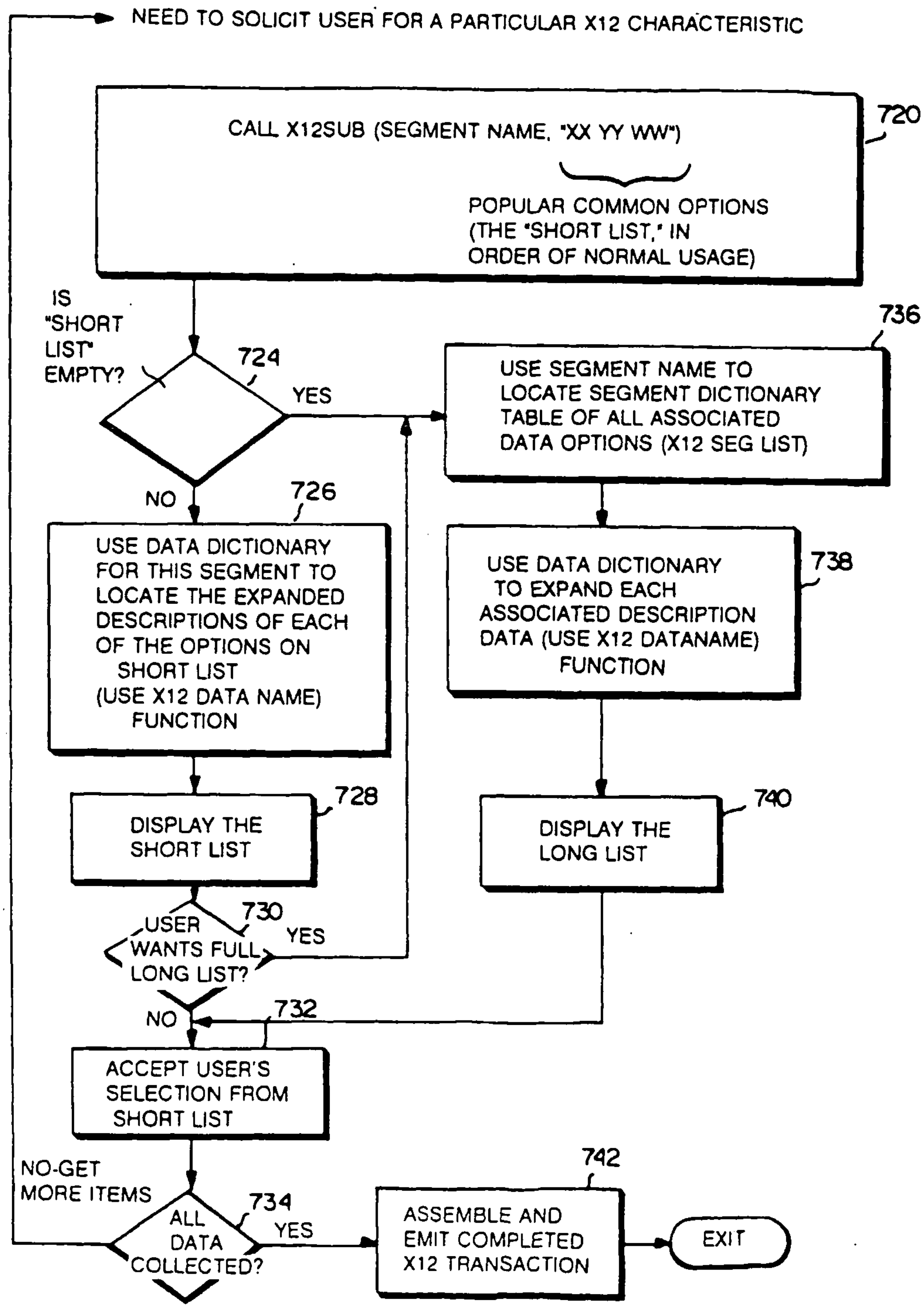

TRAVELING PROGRAM ARRIVES AT MERGING DESTINATION AND IS RUN
710
WRITE COLLECTED DATA FOR THIS INSTANCE TO SPECIAL (PERMANENT) FILE
712
HAVE ALL OTHER INSTANCES ARRIVED ?
714
YES
PROCESS COLLECTED DATA
716
NO
MSG:"WAITING FOR MORE FORMS TO ARRIVE"
720
SEND THIS FORM TO NEXT DESTINATION
718
DELETE THIS CURRENT (INSTANCE)
722
EXIT
EXIT

Fig. 39


NEED TO SOLICIT USER FOR A PARTICULAR X12 CHARACTERISTIC
CALL X12SUB (SEGMENT NAME, "XX YY WW")
POPULAR COMMON OPTIONS (THE "SHORT LIST," IN ORDER OF NORMAL USAGE)
720
IS "SHORT LIST" EMPTY?
724
YES
NO
736
USE SEGMENT NAME TO LOCATE SEGMENT DICTIONARY TABLE OF ALL ASSOCIATED DATA OPTIONS (X12 SEG LIST)
726
USE DATA DICTIONARY FOR THIS SEGMENT TO LOCATE THE EXPANDED DESCRIPTIONS OF EACH OF THE OPTIONS ON SHORT LIST (USE X12 DATA NAME) FUNCTION
USE DATA DICTIONARY TO EXPAND EACH ASSOCIATED DESCRIPTION DATA (USE X12 DATANAME) FUNCTION
738
DISPLAY THE SHORT LIST
728
DISPLAY THE LONG LIST
740
USER WANTS FULL LONG LIST?
730
YES
NO
732
ACCEPT USER'S SELECTION FROM SHORT LIST
NO-GET MORE ITEMS
ALL DATA COLLECTED?
734
YES
742
ASSEMBLE AND EMIT COMPLETED X12 TRANSACTION
EXIT

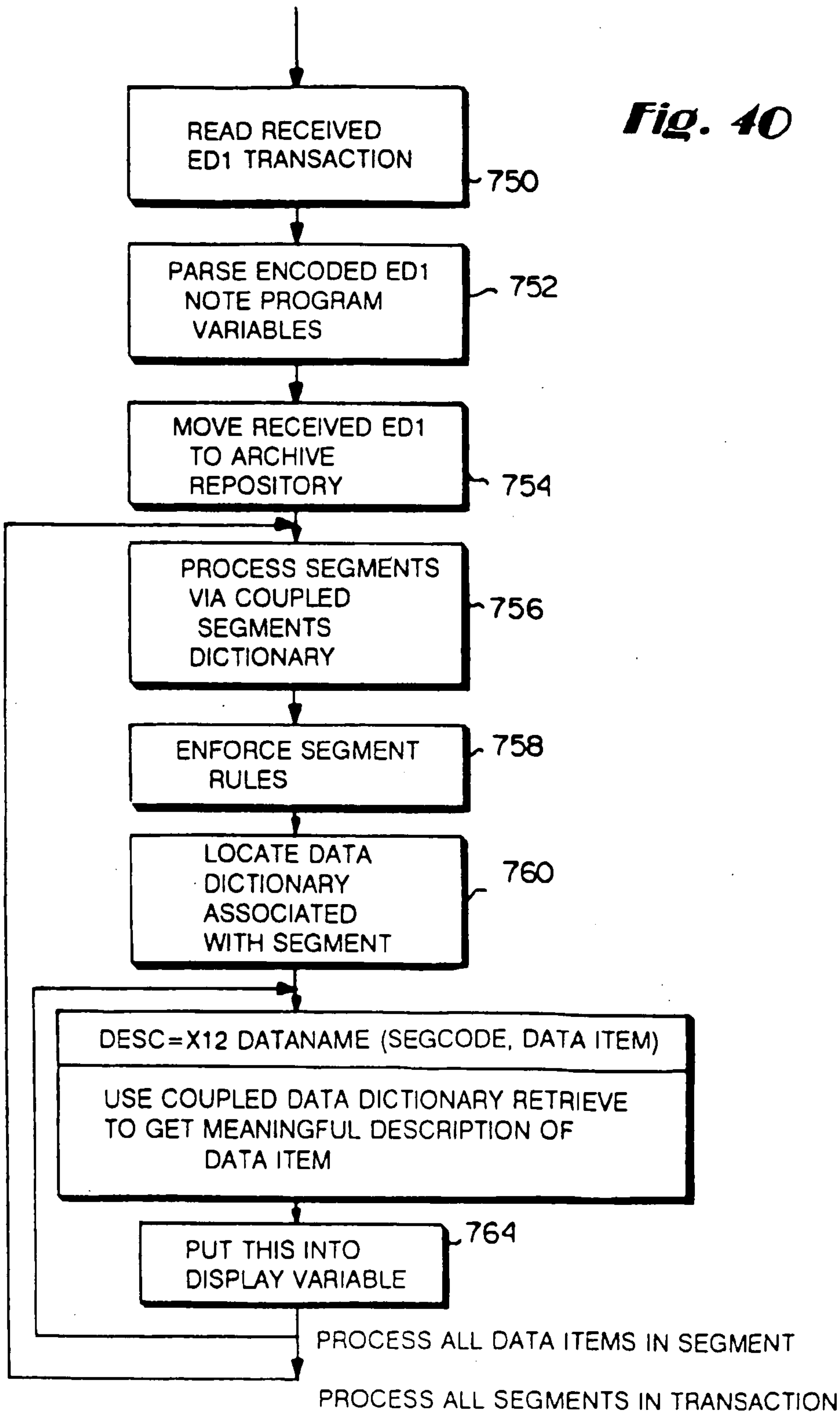

READ RECEIVED ED1 TRANSACTION
750
PARSE ENCODED ED1 NOTE PROGRAM VARIABLES
752
MOVE RECEIVED ED1 TO ARCHIVE REPOSITORY
754
PROCESS SEGMENTS VIA COUPLED SEGMENTS DICTIONARY
756
ENFORCE SEGMENT RULES
758
LOCATE DATA DICTIONARY ASSOCIATED WITH SEGMENT
760
DESC=X12 DATANAME (SEGCODE, DATA ITEM)
USE COUPLED DATA DICTIONARY RETRIEVE TO GET MEANINGFUL DESCRIPTION OF DATA ITEM
PUT THIS INTO DISPLAY VARIABLE
764
PROCESS ALL DATA ITEMS IN SEGMENT
PROCESS ALL SEGMENTS IN TRANSACTION

Fig. 40